STC micro™
宏 晶 科 技

深圳国芯人工智能有限公司

STC8A/8F 系列单片机
技术参考手册

STC MCU

技术支持网站：www.STCMCUDATA.com / ~~www.STCMCU.com~~
资料更新日期：**2020/9/23**

目录

1 概述 1
2 特性、价格及管脚 3
2.1 STC8A8K64S4A12-64Pin 系列 3
2.1.1 特性及价格 3
2.1.2 管脚图 6
2.1.3 管脚说明 10
2.2 STC8A4K64S2A12-64Pin 系列 17
2.2.1 特性及价格 17
2.2.2 管脚图 20
2.2.3 管脚说明 24
2.3 STC8F2K64S4-44Pin 系列 31
2.3.1 特性及价格 31
2.3.2 管脚图 33
2.3.3 管脚说明 36
2.4 STC8F2K64S2-44Pin 系列 40
2.4.1 特性及价格 40
2.4.2 管脚图 42
2.4.3 管脚说明 46
2.5 STC8F1K08S2-20Pin/16Pin 系列 50
2.5.1 特性及价格 50
2.5.2 管脚图 52
2.5.3 管脚说明 53
2.6 STC8F1K08-8Pin 系列 55
2.6.1 特性及价格 55
2.6.2 管脚图 56
2.6.3 管脚说明 57
3 功能脚切换 58
3.1 功能脚切换相关寄存器 58
3.1.1 总线速度控制寄存器（BUS_SPEED） 58
3.1.2 外设端口切换控制寄存器 1（P_SW1），串口 1、CCP、SPI 切换 59
3.1.3 外设端口切换控制寄存器 2（P_SW2），串口 2/3/4、I2C、比较器输出切换 59
3.1.4 时钟选择寄存器（CKSEL） 60
3.1.5 增强型 PWM 控制寄存器（PWMnCR） 60
3.2 范例程序 62
3.2.1 串口 1 切换 62
3.2.2 串口 2 切换 62
3.2.3 串口 3 切换 63
3.2.4 串口 4 切换 63
3.2.5 SPI 切换 64
3.2.6 PWM 切换 65

STC MCU

3.2.7 PCA/CCP/PWM 切换 67
3.2.8 I2C 切换 67
3.2.9 比较器输出切换 68
3.2.10 主时钟输出切换 69
4 封装尺寸图 70
4.1 SOP8 封装尺寸图 70
4.2 DFN8 封装尺寸图（3mm*3mm） 71
4.3 SOP16 封装尺寸图 72
4.4 TSSOP20 封装尺寸图 73
4.5 QFN20 封装尺寸图（3mm*3mm） 74
4.6 LQFP32 封装尺寸图（9mm*9mm） 75
4.7 QFN32 封装尺寸图（4mm*4mm） 76
4.8 LQFP48 封装尺寸图（9mm*9mm） 77
4.9 QFN48 封装尺寸图（6mm*6mm） 78
4.10 LQFP64S 封装尺寸图（12mm*12mm） 79
4.11 QFN64 封装尺寸图（8mm*8mm） 80
4.12 STC8A/8F 系列单片机命名规则 81
5 ISP 下载及典型应用线路图 82
5.1 STC8A 系列 ISP 下载应用线路图 82
5.1.1 使用 RS-232 转换器下载，也可支持仿真（使用高精度 ADC） 82
5.1.2 使用 RS-232 转换器下载，也可支持仿真（使用一般精度 ADC） 83
5.1.3 使用 PL2303-GL 下载，也可支持仿真 84
5.1.4 使用通用 USB 转串口工具下载，支持 ISP 在线下载，也可支持仿真 85
5.1.5 使用 U8-Mini 工具下载，支持 ISP 在线和脱机下载，也可支持仿真 86
5.1.6 使用 U8W 工具下载，支持 ISP 在线和脱机下载，也可支持仿真 87
5.1.7 模拟 USB 直接 ISP 下载，仅支持 ISP 下载，不支持仿真 89
5.1.8 单片机电源控制参考电路 90
5.2 STC8F 系列 ISP 下载应用线路图 91
5.2.1 使用 RS-232 转换器下载，也可支持仿真 91
5.2.2 使用 PL2303-GL 下载，也可支持仿真 92
5.2.3 使用通用 USB 转串口工具下载，支持 ISP 在线下载，也可支持仿真 93
5.2.4 使用 U8-Mini 工具下载，支持 ISP 在线和脱机下载，也可支持仿真 94
5.2.5 使用 U8W 工具下载，支持 ISP 在线和脱机下载，也可支持仿真 95
5.2.6 模拟 USB 直接 ISP 下载，仅支持 ISP 下载，不支持仿真 97
6 时钟、复位、省电模式与系统电源管理 98
6.1 系统时钟控制 98
6.1.1 系统时钟选择寄存器（CKSEL） 98
6.1.2 时钟分频寄存器（CLKDIV） 99
6.1.3 内部 24M 高精度 IRC 控制寄存器（IRC24MCR） 99
6.1.4 外部振荡器控制寄存器（XOSCCR） 99
6.1.5 内部 32KHz 低速 IRC 控制寄存器（IRC32KCR） 100
6.2 STC8A/8F 系列内部 IRC 频率调整 101
6.2.1 内部 IRC 频率调整寄存器（IRTRIM） 101
6.2.2 内部 IRC 频率微调寄存器（LIRTRIM） 102

6.2.3 时钟分频寄存器（CLKDIV） 102
6.3 STC15 系列内部 IRC 频率调整 103
6.4 系统复位 105
6.4.1 看门狗复位（WDT_CONTR） 106
6.4.2 软件复位（IAP_CONTR） 108
6.4.3 低压复位（RSTCFG） 109
6.4.4 传统 8051 高电平上电复位参考电路 110
6.5 外部晶振及外部时钟电路 111
6.5.1 外部晶振输入电路 111
6.5.2 外部时钟输入电路（P1.6 不可用作普通 I/O） 111
6.6 时钟停振/省电模式与系统电源管理 111
6.6.1 电源控制寄存器（PCON） 111
6.6.2 电压控制寄存器（VOCTRL） 112
6.7 掉电唤醒定时器 113
6.7.1 掉电唤醒定时器计数寄存器（WKTCL，WKTCH） 113
6.8 范例程序 114
6.8.1 选择系统时钟源 114
6.8.2 主时钟分频输出 115
6.8.3 看门狗定时器应用 116
6.8.4 软复位实现自定义下载 117
6.8.5 低压检测 118
6.8.6 省电模式 119
6.8.7 使用 INT0/INT1/INT2/INT3/INT4 中断唤醒 MCU 120
6.8.8 使用 T0/T1/T2/T3/T4 中断唤醒 MCU 122
6.8.9 使用 RxD/RxD2/RxD3/RxD4 中断唤醒 MCU 126
6.8.10 使用 LVD 中断唤醒 MCU 128
6.8.11 使用 CCP0/CCP1/CCP2/CCP3 中断唤醒 MCU 130
6.8.12 CMP 中断唤醒 MCU 132
6.8.13 使用 LVD 功能检测工作电压（电池电压） 133
7 存储器 137
7.1 程序存储器 137
7.2 数据存储器 139
7.2.1 内部 RAM 139
7.2.2 程序状态寄存器（PSW） 140
7.2.3 内部扩展 RAM，XRAM，XDATA 140
7.2.4 辅助寄存器（AUXR） 141
7.2.5 外部扩展 RAM，XRAM，XDATA 142
7.2.6 总线速度控制寄存器（BUS_SPEED） 142
7.2.7 8051 中可位寻址的数据存储器 144
7.3 存储器中的特殊参数，在 ISP 下载时可烧录进程序 FLASH 146
7.3.1 读取内部 1.344V 参考信号源值 (从 Flash 程序存储器（ROM）中读取) 147
~~7.3.2 读取内部 1.344V 参考信号源值 (从 RAM 中读取)~~ 150
7.3.3 读取全球唯一 ID 号 (从 Flash 程序存储器（ROM）中读取) 152
~~7.3.4 读取全球唯一 ID 号 (从 RAM 中读取)~~ 154

7.3.5 读取 32K 掉电唤醒定时器的频率 (从 Flash 程序存储器（ROM）中读取) 156
~~7.3.6 读取 32K 掉电唤醒定时器的频率 (从 RAM 中读取)~~ 158
7.3.7 用户自定义内部 IRC 频率 (从 Flash 程序存储器（ROM）中读取) 160
~~7.3.8 用户自定义内部 IRC 频率 (从 RAM 中读取)~~ 162
8 特殊功能寄存器 164
8.1 STC8A8K64S4A12-64Pin 系列 164
8.2 STC8A4K64S2A12-64Pin 系列 165
8.3 STC8F2K64S4-44Pin 系列 166
8.4 STC8F2K64S2-44Pin 系列 167
8.5 STC8F1K08S2-20Pin/16Pin 系列 168
8.6 STC8F1K08S-8Pin 系列 169
8.7 特殊功能寄存器列表 170
9 I/O 口 175
9.1 I/O 口相关寄存器 175
9.1.1 端口数据寄存器（Px） 176
9.1.2 端口模式配置寄存器（PxM0，PxM1） 176
9.1.3 端口上拉电阻控制寄存器（PxPU） 177
9.1.4 端口施密特触发控制寄存器（PxNCS） 177
9.2 配置 I/O 口 179
9.3 I/O 的结构图 181
9.3.1 准双向口（弱上拉） 181
9.3.2 推挽输出 181
9.3.3 高阻输入 182
9.3.4 开漏输出 182
9.3.5 新增 4.1K 上拉电阻 183
9.4 特殊 I/O 口说明 184
9.4.1 P2.0/RSTCV 184
9.4.2 PWM 相关 I/O 口 184
9.5 范例程序 185
9.5.1 端口模式设置 185
9.5.2 双向口读写操作 186
9.6 一种典型三极管控制电路 189
9.7 典型发光二极管控制电路 189
9.8 混合电压供电系统 3V/5V 器件 I/O 口互连 190
9.9 如何让 I/O 口上电复位时为低电平 191
9.10 利用 74HC595 驱动 8 个数码管(串行扩展,3 根线)的线路图 192
9.11 I/O 口直接驱动 LED 数码管应用线路图 193
9.12 用 STC 系列 MCU 的 I/O 口直接驱动段码 LCD 194
10 指令系统 213
11 中断系统 217
11.1 STC8G 系列中断源 217
11.2 STC8A/8F 系列中断结构图 219
11.3 STC8A/8F 系列中断列表 220
11.4 中断相关寄存器 223

STCMCU

11.4.1 中断使能寄存器（中断允许位） 224
11.4.2 中断请求寄存器（中断标志位） 227
11.4.3 中断优先级寄存器 230
11.5 范例程序 232
11.5.1 INT0 中断（上升沿和下降沿） 232
11.5.2 INT0 中断（下降沿） 233
11.5.3 INT1 中断（上升沿和下降沿） 233
11.5.4 INT1 中断（下降沿） 234
11.5.5 INT2 中断（下降沿） 235
11.5.6 INT3 中断（下降沿） 236
11.5.7 INT4 中断（下降沿） 237
11.5.8 定时器 0 中断 238
11.5.9 定时器 1 中断 239
11.5.10 定时器 2 中断 240
11.5.11 定时器 3 中断 241
11.5.12 定时器 4 中断 242
11.5.13 UART1 中断 243
11.5.14 UART2 中断 245
11.5.15 UART3 中断 247
11.5.16 UART4 中断 248
11.5.17 ADC 中断 250
11.5.18 LVD 中断 251
11.5.19 PCA 中断 252
11.5.20 SPI 中断 254
11.5.21 CMP 中断 255
11.5.22 PWM 中断 256
11.5.23 I2C 中断 259
12 定时器/计数器 262
12.1 定时器的相关寄存器 262
12.2 定时器 0/1 264
12.2.1 定时器 0/1 控制寄存器（TCON） 264
12.2.2 定时器 0/1 模式寄存器（TMOD） 264
12.2.3 定时器 0 模式 0（16 位自动重装载模式） 265
12.2.4 定时器 0 模式 1（16 位不可重装载模式） 266
12.2.5 定时器 0 模式 2（8 位自动重装载模式） 267
12.2.6 定时器 0 模式 3（不可屏蔽中断 16 位自动重装载，实时操作系统节拍器） 267
12.2.7 定时器 1 模式 0（16 位自动重装载模式） 268
12.2.8 定时器 1 模式 1（16 位不可重装载模式） 269
12.2.9 定时器 1 模式 2（8 位自动重装载模式） 270
12.2.10 定时器 0 计数寄存器（TL0，TH0） 270
12.2.11 定时器 1 计数寄存器（TL1，TH1） 270
12.2.12 辅助寄存器 1（AUXR） 270
12.2.13 中断与时钟输出控制寄存器（INTCLKO） 271
12.2.14 定时器 0 定时计算公式 271

STCMCU

12.2.15 定时器 1 定时计算公式 272
12.3 定时器 2 273
12.3.1 辅助寄存器 1（AUXR） 273
12.3.2 中断与时钟输出控制寄存器（INTCLKO） 273
12.3.3 定时器 2 计数寄存器（T2L，T2H） 273
12.3.4 定时器 2 工作模式 274
12.3.5 定时器 2 计算公式 274
12.4 定时器 3/4 275
12.4.1 定时器 4/3 控制寄存器（T4T3M） 275
12.4.2 定时器 3 计数寄存器（T3L，T3H） 275
12.4.3 定时器 4 计数寄存器（T4L，T4H） 275
12.4.4 定时器 3 工作模式 276
12.4.5 定时器 4 工作模式 277
12.4.6 定时器 3 计算公式 277
12.4.7 定时器 4 计算公式 278
12.5 范例程序 279
12.5.1 定时器 0（模式 0－16 位自动重载），用作定时 279
12.5.2 定时器 0（模式 1－16 位不自动重载），用作定时 280
12.5.3 定时器 0（模式 2－8 位自动重载），用作定时 282
12.5.4 定时器 0（模式 3－16 位自动重载不可屏蔽中断），用作定时 284
12.5.5 定时器 0（外部计数－扩展 T0 为外部下降沿中断） 286
12.5.6 定时器 0（测量脉宽－INT0 高电平宽度） 288
12.5.7 定时器 0（模式 0），时钟分频输出 290
12.5.8 定时器 1（模式 0－16 位自动重载），用作定时 292
12.5.9 定时器 1（模式 1－16 位不自动重载），用作定时 294
12.5.10 定时器 1（模式 2－8 位自动重载），用作定时 295
12.5.11 定时器 1（外部计数－扩展 T1 为外部下降沿中断） 297
12.5.12 定时器 1（测量脉宽－INT1 高电平宽度） 299
12.5.13 定时器 1（模式 0），时钟分频输出 301
12.5.14 定时器 1（模式 0）做串口 1 波特率发生器 303
12.5.15 定时器 1（模式 2）做串口 1 波特率发生器 307
12.5.16 定时器 2（16 位自动重载），用作定时 311
12.5.17 定时器 2（外部计数－扩展 T2 为外部下降沿中断） 313
12.5.18 定时器 2，时钟分频输出 315
12.5.19 定时器 2 做串口 1 波特率发生器 317
12.5.20 定时器 2 做串口 2 波特率发生器 320
12.5.21 定时器 2 做串口 3 波特率发生器 324
12.5.22 定时器 2 做串口 4 波特率发生器 328
12.5.23 定时器 3（16 位自动重载），用作定时 332
12.5.24 定时器 3（外部计数－扩展 T3 为外部下降沿中断） 335
12.5.25 定时器 3，时钟分频输出 337
12.5.26 定时器 3 做串口 3 波特率发生器 339
12.5.27 定时器 4（16 位自动重载），用作定时 343
12.5.28 定时器 4（外部计数－扩展 T4 为外部下降沿中断） 346

12.5.29 定时器 4，时钟分频输出 348
12.5.30 定时器 4 做串口 4 波特率发生器 350
13 串口通信 355
13.1 串口相关寄存器 355
13.2 串口 1 356
13.2.1 串口 1 控制寄存器（SCON） 356
13.2.2 串口 1 数据寄存器（SBUF） 356
13.2.3 电源管理寄存器（PCON） 357
13.2.4 辅助寄存器 1（AUXR） 357
13.2.5 串口 1 模式 0，模式 0 波特率计算公式 357
13.2.6 串口 1 模式 1，模式 1 波特率计算公式 358
13.2.7 串口 1 模式 2，模式 2 波特率计算公式 361
13.2.8 串口 1 模式 3，模式 3 波特率计算公式 361
13.2.9 自动地址识别 362
13.2.10 串口 1 从机地址控制寄存器（SADDR，SADEN） 362
13.3 串口 2 364
13.3.1 串口 2 控制寄存器（S2CON） 364
13.3.2 串口 2 数据寄存器（S2BUF） 364
13.3.3 串口 2 模式 0，模式 0 波特率计算公式 364
13.3.4 串口 2 模式 1，模式 1 波特率计算公式 365
13.4 串口 3 367
13.4.1 串口 3 控制寄存器（S3CON） 367
13.4.2 串口 3 数据寄存器（S3BUF） 367
13.4.3 串口 3 模式 0，模式 0 波特率计算公式 367
13.4.4 串口 3 模式 1，模式 1 波特率计算公式 368
13.5 串口 4 370
13.5.1 串口 4 控制寄存器（S4CON） 370
13.5.2 串口 4 数据寄存器（S4BUF） 370
13.5.3 串口 4 模式 0，模式 0 波特率计算公式 370
13.5.4 串口 4 模式 1，模式 1 波特率计算公式 371
13.6 串口注意事项 373
13.7 范例程序 374
13.7.1 串口 1 使用定时器 2 做波特率发生器 374
13.7.2 串口 1 使用定时器 1（模式 0）做波特率发生器 377
13.7.3 串口 1 使用定时器 1（模式 2）做波特率发生器 381
13.7.4 串口 2 使用定时器 2 做波特率发生器 385
13.7.5 串口 3 使用定时器 2 做波特率发生器 389
13.7.6 串口 3 使用定时器 3 做波特率发生器 393
13.7.7 串口 4 使用定时器 2 做波特率发生器 397
13.7.8 串口 4 使用定时器 4 做波特率发生器 401
13.7.9 串口多机通讯 406
14 比较器，掉电检测，内部 1.344V 参考信号源 407
14.1 比较器内部结构图 407
14.2 比较器相关的寄存器 408

14.2.1 比较器控制寄存器 1（CMPCR1） 408
14.2.2 比较器控制寄存器 2（CMPCR2） 409
14.3 范例程序 410
14.3.1 比较器的使用（中断方式） 410
14.3.2 比较器的使用（查询方式） 412
14.3.3 比较器的多路复用应用（比较器+ADC 输入通道） 415
14.3.4 比较器作外部掉电检测（掉电过程中应及时保存用户数据到 EEPROM 中） 417
14.3.5 比较器检测工作电压（电池电压） 418
15 IAP/EEPROM/DATA-FLASH 423
15.1 EEPROM 操作时间 423
15.2 EEPROM 相关的寄存器 423
15.2.1 EEPROM 数据寄存器（IAP_DATA） 423
15.2.2 EEPROM 地址寄存器（IAP_ADDR） 424
15.2.3 EEPROM 命令寄存器（IAP_CMD） 424
15.2.4 EEPROM 触发寄存器（IAP_TRIG） 424
15.2.5 EEPROM 控制寄存器（IAP_CONTR） 424
15.3 关于 EEPROM 编程和擦除等待时间的重要说明 426
15.4 EEPROM 大小及地址 427
15.5 范例程序 430
15.5.1 EEPROM 基本操作 430
15.5.2 使用 MOVC 读取 EEPROM 433
15.5.3 使用串口送出 EEPROM 数据 436
16 ADC 模数转换，内部 1.344V 参考信号源 440
16.1 ADC 相关的寄存器 440
16.1.1 ADC 控制寄存器（ADC_CONTR） 441
16.1.2 ADC 配置寄存器（ADCCFG） 442
16.1.3 ADC 转换结果寄存器（ADC_RES，ADC_RESL） 443
16.2 ADC 相关计算公式 444
16.2.1 ADC 转换结果计算公式 444
16.2.2 反推 ADC 输入电压计算公式 444
16.2.3 反推工作电压计算公式 444
16.3 12 位 ADC 静态特性 444
16.4 ADC 应用参考线路图 445
16.4.1 高精度 ADC 应用 445
16.4.2 ADC 一般应用（对 ADC 精度要求不高的应用） 446
16.5 范例程序 447
16.5.1 ADC 基本操作（查询方式） 447
16.5.2 ADC 基本操作（中断方式） 448
16.5.3 格式化 ADC 转换结果 449
16.5.4 利用 ADC 第 15 通道测量外部电压或电池电压 451
16.5.5 ADC 做电容感应触摸按键 453
16.5.6 ADC 作按键扫描应用线路图 466
16.5.7 检测负电压参考线路图 467
16.5.8 常用加法电路在 ADC 中的应用 468

17 PCA/CCP/PWM 应用 469
17.1 PCA 相关的寄存器 469
17.1.1 PCA 控制寄存器（CCON） 470
17.1.2 PCA 模式寄存器（CMOD） 470
17.1.3 PCA 计数器寄存器（CL，CH） 470
17.1.4 PCA 模块模式控制寄存器（CCAPMn） 471
17.1.5 PCA 模块模式捕获值/比较值寄存器（CCAPnL，CCAPnH） 471
17.1.6 PCA 模块 PWM 模式控制寄存器（PCA_PWMn） 471
17.2 PCA 工作模式 473
17.2.1 捕获模式 473
17.2.2 软件定时器模式 473
17.2.3 高速脉冲输出模式 474
17.2.4 PWM 脉宽调制模式及频率计算公式 474
17.3 利用 CCP/PCA/PWM 模块实现 8~16 位 DAC 的参考线路图 479
17.4 范例程序 480
17.4.1 PCA 输出 PWM（6/7/8/10 位） 480
17.4.2 PCA 捕获测量脉冲宽度 482
17.4.3 PCA 实现 16 位软件定时 485
17.4.4 PCA 输出高速脉冲 487
17.4.5 PCA 扩展外部中断 490
18 精度可达 15 位的增强型 PWM 493
18.1 PWM 相关的寄存器 493
18.1.1 增强型 PWM 配置寄存器（PWMCFG） 495
18.1.2 PWM 中断标志寄存器（PWMIF） 495
18.1.3 PWM 异常检测控制寄存器（PWMFDCR） 495
18.1.4 PWM 控制寄存器（PWMCR） 496
18.1.5 PWM 计数器寄存器（PWMCH，PWMCL） 496
18.1.6 PWM 时钟选择寄存器（PWMCKS），输出频率计算公式 496
18.1.7 PWM 触发 ADC 计数器寄存器（PWMnTADC） 498
18.1.8 PWM 电平输出设置计数值寄存器（PWMnT1，PWMnT2） 498
18.1.9 PWM 通道控制寄存器（PWMnCR） 499
18.1.10 PWM 通道电平保持控制寄存器（PWMnHLD） 500
18.2 范例程序 501
18.2.1 输出任意周期和任意占空比的波形 501
18.2.2 两路 PWM 实现互补对称带死区控制的波形 503
18.2.3 PWM 实现渐变灯（呼吸灯） 507
19 同步串行外设接口 SPI 512
19.1 SPI 相关的寄存器 512
19.1.1 SPI 状态寄存器（SPSTAT） 512
19.1.2 SPI 控制寄存器（SPCTL），SPI 速度控制 512
19.1.3 SPI 数据寄存器（SPDAT） 513
19.2 SPI 通信方式 514
19.2.1 单主单从 514
19.2.2 互为主从 514

19.2.3 单主多从 515
19.3 配置 SPI 516
19.4 数据模式 518
19.5 范例程序 520
19.5.1 SPI 单主单从系统主机程序（中断方式） 520
19.5.2 SPI 单主单从系统从机程序（中断方式） 522
19.5.3 SPI 单主单从系统主机程序（查询方式） 524
19.5.4 SPI 单主单从系统从机程序（查询方式） 526
19.5.5 SPI 互为主从系统程序（中断方式） 529
19.5.6 SPI 互为主从系统程序（查询方式） 531
20 I²C 总线 535
20.1 I²C 相关的寄存器 535
20.2 I²C 主机模式 536
20.2.1 I2C 配置寄存器（I2CCFG），总线速度控制 536
20.2.2 I2C 主机控制寄存器（I2CMSCR） 537
20.2.3 I2C 主机辅助控制寄存器（I2CMSAUX） 538
20.2.4 I2C 主机状态寄存器（I2CMSST） 538
20.3 I²C 从机模式 540
20.3.1 I2C 从机控制寄存器（I2CSLCR） 540
20.3.2 I2C 从机状态寄存器（I2CSLST） 540
20.3.3 I2C 从机地址寄存器（I2CSLADR） 542
20.3.4 I2C 数据寄存器（I2CTXD，I2CRXD） 543
20.4 范例程序 544
20.4.1 I²C 主机模式访问 AT24C256（中断方式） 544
20.4.2 I²C 主机模式访问 AT24C256（查询方式） 550
20.4.3 I²C 主机模式访问 PCF8563 556
20.4.4 I²C 从机模式（中断方式） 561
20.4.5 I²C 从机模式（查询方式） 566
20.4.6 测试 I²C 从机模式代码的主机代码 570
21 增强型双数据指针 577
21.1 相关的特殊功能寄存器 577
21.1.1 第 1 组 16 位数据指针寄存器（DPTR0） 577
21.1.2 第 2 组 16 位数据指针寄存器（DPTR1） 577
21.1.3 数据指针控制寄存器（DPS） 577
21.1.4 数据指针控制寄存器（TA） 578
21.2 范例程序 580
21.2.1 示例代码 1 580
21.2.2 示例代码 2 581
附录 A 编译器（汇编器）/仿真器使用指南 583
附录 B 如何让传统的 8051 单片机学习板可仿真 590
附录 C STC-USB 驱动程序安装说明 592
附录 D USB 下载步骤演示 655
附录 E RS485 自动控制或 I/O 口控制线路图 659
附录 F STC 工具使用说明书 660

F.1 概述 660
F.2 系统可编程（ISP）流程说明 660
F.3 USB 型联机/脱机下载工具 U8W/U8W-Mini 661
F.3.1 安装 U8W/U8W-Mini 驱动程序 663
F.3.2 U8W 的功能介绍 666
F.3.3 U8W 的在线联机下载使用说明 667
F.3.4 U8W 的脱机下载使用说明 670
F.3.5 U8W-Mini 的功能介绍 678
F.3.6 U8W-Mini 的在线联机下载使用说明 679
F.3.7 U8W-Mini 的脱机下载使用说明 680
F.3.8 制作/更新 U8W/U8W-Mini 686
F.3.9 U8W/U8W-Mini 设置直通模式（可用于仿真） 688
F.3.10 U8W/U8W-Mini 的参考电路 688
F.4 STC 通用 USB 转串口工具 690
F.4.1 STC 通用 USB 转串口工具外观图 690
F.4.2 STC 通用 USB 转串口工具布局图 691
F.4.3 STC 通用 USB 转串口工具驱动安装 692
F.4.4 使用 STC 通用 USB 转串口工具下载程序到 MCU 693
F.4.5 使用 STC 通用 USB 转串口工具仿真用户代码 695
F.5 应用线路图 702
F.5.1 U8W 工具应用参考线路图 702
F.5.2 STC 通用 USB 转串口工具应用参考线路图 702
附录 G U8W 下载工具中 RS485 部分线路图 704
附录 H 运行用户程序时收到用户命令后自动启动 ISP 下载(不停电) 705
附录 I 使用 STC 的 IAP 系列单片机开发自己的 ISP 程序 707
附录 J 用户程序复位到系统区进行 ISP 下载的方法（不停电） 719
附录 K 使用第三方 MCU 对 STC8A/8F 系列单片机进行 ISP 下载范例程序 725
附录 L 使用第三方应用程序调用 STC 发布项目程序对单片机进行 ISP 下载 733
附录 M 在 Keil 中建立多文件项目的方法 737
附录 N 电气特性 741
N.1 绝对最大额定值 741
N.2 直流特性（STC8A 系列） 742
N.3 直流特性（STC8F 系列） 743
N.4 内部 IRC 温漂特性（参考温度 25℃） 744
附录 O 应用注意事项 745
O.1 关于 STC8A 系列芯片 ADC 的问题 745
O.2 关于 STC8A/8F 系列芯片问题总结 746
O.3 关于使用 CLR 指令关闭 EA 的重要说明 747
O.4 关于 EEPROM 编程和擦除等待时间的重要说明 748
O.5 STC8F2K64S4 系列应用注意事项 750
O.6 STC8F2K64S2 系列应用注意事项 751
O.7 STC8A8K64S4A12 系列应用注意事项 752
O.8 STC8A4K64S2A12 系列应用注意事项 753
O.9 使用外部晶振对 STC8 系列进行仿真的注意事项 754

附录 P QFN/DFN 封装元器件焊接方法 755
附录 Q 更新记录 758
附录 R 官方网址说明 763

STC MCU

1 概述

STC8 系列单片机是不需要外部晶振和外部复位的单片机，是以超强抗干扰/超低价/高速/低功耗为目标的 8051 单片机，在相同的工作频率下，STC8 系列单片机比传统的 8051 约快 12 倍（速度快 11.2~13.2 倍），依次按顺序执行完全部的 111 条指令，STC8 系列单片机仅需 147 个时钟，而传统 8051 则需要 1944 个时钟。STC8 系列单片机是 STC 生产的单时钟/机器周期(1T)的单片机，是宽电压/高速/高可靠/低功耗/强抗静电/较强抗干扰的新一代 8051 单片机，超级加密。指令代码完全兼容传统 8051。

MCU 内部集成高精度 R/C 时钟(±0.3%，常温下+25℃)，-1.8%～+0.8%温飘(-40℃~+85℃)，-1.0%～+0.5%温飘(-20℃~+65℃)。ISP 编程时 5MHz~30MHz 宽范围可设置，可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP 编程时 4 级复位门槛电压可选)。

MCU 内部有 3 个可选时钟源：内部 24MHz 高精度 IRC 时钟（可适当调高或调低）、内部 32KHz 的低速 IRC、外部 4M~33M 晶振或外部时钟信号。用户代码中可自由选择时钟源，时钟源选定后可再经过 8-bit 的分频器分频后再将时钟信号提供给 CPU 和各个外设（如定时器、串口、SPI 等）。

MCU 提供两种低功耗模式：IDLE 模式和 STOP 模式。IDLE 模式下，MCU 停止给 CPU 提供时钟，CPU 无时钟，CPU 停止执行指令，但所有的外设仍处于工作状态，此时功耗约为 1.3mA（6MHz 工作频率）。STOP 模式即为主时钟停振模式，即传统的掉电模式/停电模式/停机模式，此时 CPU 和全部外设都停止工作，功耗可降低到 0.1uA 以下。

IDLE 模式可以由外部中断（INT0/INT1/INT2/INT3/INT4）、定时器中断（定时器 0/定时器 1/定时器 2/定时器 3/定时器 4）、串口中断（串口 1/串口 2/串口 3/串口 4）、PCA/CCP/PWM 中断、增强型 PWM、增强型 PWM 异常检测、ADC 模数转换中断、LVD 低压检测、SPI 中断、I2C 中断、比较器中断唤醒。

STOP 模式可以由 INT0/INT1 端口上升沿/下降沿、INT2/INT3/INT4 端口下降沿、T0/T1/T2/T3/T4 端口下降沿、RxD/RxD_2/RxD_3/RxD_4 端口下降沿、RxD2/RxD2_2 端口下降沿、RxD2/RxD2_2 端口下降沿、RxD3/RxD3_2 端口下降沿、RxD4/RxD4_2 端口下降沿、CCP0/CCP0_2/CCP0_3/CCP0_4 端口下降沿、CCP1/CCP1_2/CCP1_3/CCP1_4 端口下降沿、CCP2/CCP2_2/CCP2_3/CCP2_4 端口下降沿、CCP3/CCP3_2/CCP3_3/CCP3_4 端口下降沿、LVD 低压检测以及掉电唤醒定时器（进入掉电模式前需要先使能掉电唤醒定时器）唤醒。

MCU 提供了丰富的数字外设（4 个串口、5 个定时器、4 组 PCA、8 组增强型 PWM 以及 I^2C、SPI）接口与模拟外设（速度高达 800K 即每秒 80 万次采样的 12 位*15 路超高速 ADC、比较器），可满足广大用户的设计需求。

数字功能可使用程序在多个管脚之间进行切换。串口 1 可以组为单位（[TxD/RxD]为一组）在[P3.0/P3.1]、[P3.6/P3.7]、[P1.6/P1.7]、[P4.3/P4.4]这 4 组之间进行任意切换；串口 2 可以组为单位（[TxD2/RxD2]为一组）在[P1.0/P1.1]、[P4.0/P4.2]这 2 组之间进行任意切换；串口 3 可以组为单位（[TxD3/RxD3]为一组）在[P0.0/P0.1]、[P5.0/P5.1]这 2 组之间进行任意切换；串口 4 可以组为单位（[TxD4/RxD4]为一组）在[P0.2/P0.3]、[P5.2/P5.3]这 2 组之间进行任意切换；PCA 可以组为单位（[ECI/CCP0/CCP1/CCP2/CCP3]为一组）在[P1.2/P1.7/P1.6/P1.5/P1.4]、[P2.2/P2.3/P2.4/P2.5/P2.6]、[P7.4/P7.0/P7.1/P7.2/P7.3]、[P3.5/P3.3/P3.2/P3.1/P3.0]这 4 组之间进行任意切换；SPI 可以组为单位（[SS/MOSI/MISO/SCLK]为一组）在[P1.2/P1.3/P1.4/P1.5]、[P2.2/P2.3/P2.4/P2.5]、[P7.4/P7.5/P7.6/P7.7]、[P3.5/P3.4/P3.3/P3.2]这 4 组之间进行任意切换；I^2C 可以组为单位（[SCL/SDA]为一组）在[P1.5/P1.4]、[P2.5/P2.4]、[P7.7/P7.6]、[P3.2/P3.3]这 4 组之间进行任意切换；增强型 PWM 的每一路均可各自独立地在 3 个端口之间任意切换（详细切换介绍请参考第 3.3 章的功能脚切换）。

STC8 系列单片机内部集成了增强型的双数据指针。通过程序控制, 可实现数据指针自动递增或递减功能以及两组数据指针的自动切换功能。

产品线	I/O	UART	定时器	ADC	增强型 PWM	PCA	比较器	SPI	I2C
STC8A8K64S4A12-64Pin 系列	59	4	5	15_{CH}*12_{B}	●	●	●	●	●
STC8A4K64S2A12-64Pin 系列	59	2	5	15_{CH}*12_{B}	●	●	●	●	●
STC8F2K64S4-44Pin 系列	42	4	5				●	●	●
STC8F2K64S2-44Pin 系列	42	2	5				●	●	●
STC8F1K08S2-20Pin/16Pin 系列	18	2	3				●	●	●
STC8F1K08-8Pin 系列	6	1	3					●	●

STC MCU

2 特性、价格及管脚

2.1 STC8A8K64S4A12-64Pin系列

2.1.1 特性及价格

- 选型价格（不需要外部晶振、不需要外部复位，12 位 ADC，15 通道）

单片机型号	工作电压（V）	Flash 程序存储器 10 万次 字节	大容量扩展 SRAM 字节	强大的双 DPTR 可增可减	EEPROM 10 万次 字节	I/O 口最多数量	串口并可掉电唤醒	SPI	I^2C	定时器/计数器（T0-T4 外部管脚也可掉电唤醒）	16 位高级 PWM 定时器 互补对称死区	15 位增强型 PWM（带死区控制）	PCA/CCP/PWM（可当外部中断并可掉电唤醒）	掉电唤醒专用定时器	15 路高速 ADC(8 路 PWM 可当 8 路 D/A 使用)	比较器（可当 1 路 A/D，可作外部掉电检测）	内部低压检测中断并可掉电唤醒	看门狗 复位定时器	内部高可靠复位（可选复位门槛电压）	内部高精准时钟（24MHz 可调）	可对外输出时钟及复位	程序加密后传输（防拦截）	可设置下次更新程序需口令	支持 RS485 下载	支持 USB 直接下载	本身就可在线仿真	封装						2017 年新品供货信息
																											LQFP64S	QFN64 <8x8mm>	LQFP48	QFN48 <6x6mm>	LQFP44	PDIP40	
STC8A8K16S4A12	2.0-5.5	16K	8K	2	48K	59	4	有	有	5	-	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	¥3.4		¥3.2		¥3.2		大量供货
STC8A8K32S4A12	2.0-5.5	32K	8K	2	32K	59	4	有	有	5	-	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	¥3.6		¥3.3		¥3.3		
STC8A8K60S4A12	2.0-5.5	60K	8K	2	4K	59	4	有	有	5	-	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	¥3.8		¥3.4		¥3.4		
STC8A8K64S4A12	2.0-5.5	64K	8K	2	IAP	59	4	有	有	5	-	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	¥3.8		¥3.4		¥3.4		

注：以上的单价为 10K 及以上订货量的价格，量小则每片需增加 0.1 元人民币。当订货的总额达到或高于 3000 元时，可免运费发货，否则需要由客户承担运费。零售 10 片起售。

- **内核**
  - ✓ 超高速 8051 内核（1T），比传统 8051 约快 12 倍以上
  - ✓ 指令代码完全兼容传统 8051
  - ✓ 22 个中断源，4 级中断优先级
  - ✓ 支持在线仿真
- **工作电压**
  - ✓ 2.0V～5.5V
  - ✓ 内建 LDO
- **工作温度**
  - ✓ -40℃～85℃（超温度范围应用请参考电器特性章节说明）
- **Flash 存储器**
  - ✓ 最大 64K 字节 FLASH 程序存储器（ROM），用于存储用户代码
  - ✓ 支持用户配置 EEPROM 大小，512 字节单页擦除，擦写次数可达 10 万次以上
  - ✓ 支持在系统编程方式（ISP）更新用户应用程序，无需专用编程器
  - ✓ 支持单芯片仿真，无需专用仿真器，理论断点个数无限制
- **SRAM**
  - ✓ 128 字节内部直接访问 RAM（DATA）

- ✓ 128 字节内部间接访问 RAM（IDATA）
- ✓ 8192 字节内部扩展 RAM（内部 XDATA）
- ✓ 外部最大可扩展 64K 字节 RAM（外部 XDATA）

➢ **时钟控制**

- ✓ 内部 24MHz 高精度 IRC（ISP 编程时可进行上下调整）
  - ⌖ 误差±0.3%（常温下 25℃）
  - ⌖ -1.8%～+0.8%温漂（全温度范围，-40℃～85℃）
  - ⌖ -1.0%～+0.5%温漂（温度范围，-20℃～65℃）
- ✓ 内部 32KHz 低速 IRC（误差较大）
- ✓ 外部晶振（4MHz～33MHz）和外部时钟
  用户可自由选择上面的 3 种时钟源

➢ **复位**

- ✓ 硬件复位
  - ⌖ 上电复位
  - ⌖ 复位脚复位（高电平复位），出厂时 P5.4 默认为 IO 口，ISP 下载时可将 P5.4 管脚设置为复位脚
  - ⌖ 看门狗溢出复位
  - ⌖ 低压检测复位，提供 4 级低压检测电压：2.2V、2.4V、2.7V、3.0V
- ✓ 软件复位
  - ⌖ 软件方式写复位触发寄存器

➢ **中断**

- ✓ 提供 22 个中断源：INT0、INT1、INT2、INT3、INT4、定时器 0、定时器 1、定时器 2、定时器 3、定时器 4、串口 1、串口 2、串口 3、串口 4、ADC 模数转换、LVD 低压检测、SPI、I^2C、比较器、PCA/CCP/PWM、增强型 PWM、增强型 PWM 异常检测
- ✓ 提供 4 级中断优先级

➢ **数字外设**

- ✓ 5 个 16 位定时器：定时器 0、定时器 1、定时器 2、定时器 3、定时器 4，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式
- ✓ 4 个高速串口：串口 1、串口 2、串口 3、串口 4，波特率时钟源最快可为 FOSC/4
- ✓ 4 组 16 位 PCA 模块：CCP0、CCP1、CCP2、CCP3，可用于捕获、高速脉冲输出，及 6/7/8/10 位的 PWM 输出
- ✓ 8 组 15 位增强型 PWM，可实现带死区的控制信号，并支持外部异常检测功能，另外还有 4 组传统的 PCA/CCP/PWM 可作 PWM
- ✓ SPI：支持主机模式和从机模式以及主机/从机自动切换
- ✓ I^2C：支持主机模式和从机模式

➢ **模拟外设**

- ✓ 超高速 ADC，支持 **12 位精度** 15 通道（通道 0～通道 14）的模数转换，**速度最快能达到 800K（每秒进行 80 万次 ADC 转换）**
- ✓ ADC 的通道 15 用于测试内部 1.344V 参考信号源（芯片在出厂时，内部参考信号源已调整为 1.344V）
- ✓ 比较器，一组比较器（比较器的正端可选择 CMP+端口和所有的 ADC 输入端口，所以比较器可当作多路比较器进行分时复用）
- ✓ DAC：4 路 PCA/CCP/PWM 可当 4 路 DAC 使用

➢ **GPIO**

- ✓ 最多可达 59 个 GPIO：P0.0~P0.7、P1.0~P1.7、P2.0~P2.7、P3.0~P3.7、P4.0~P4.4、P5.0~P5.5、P6.0~P6.7、

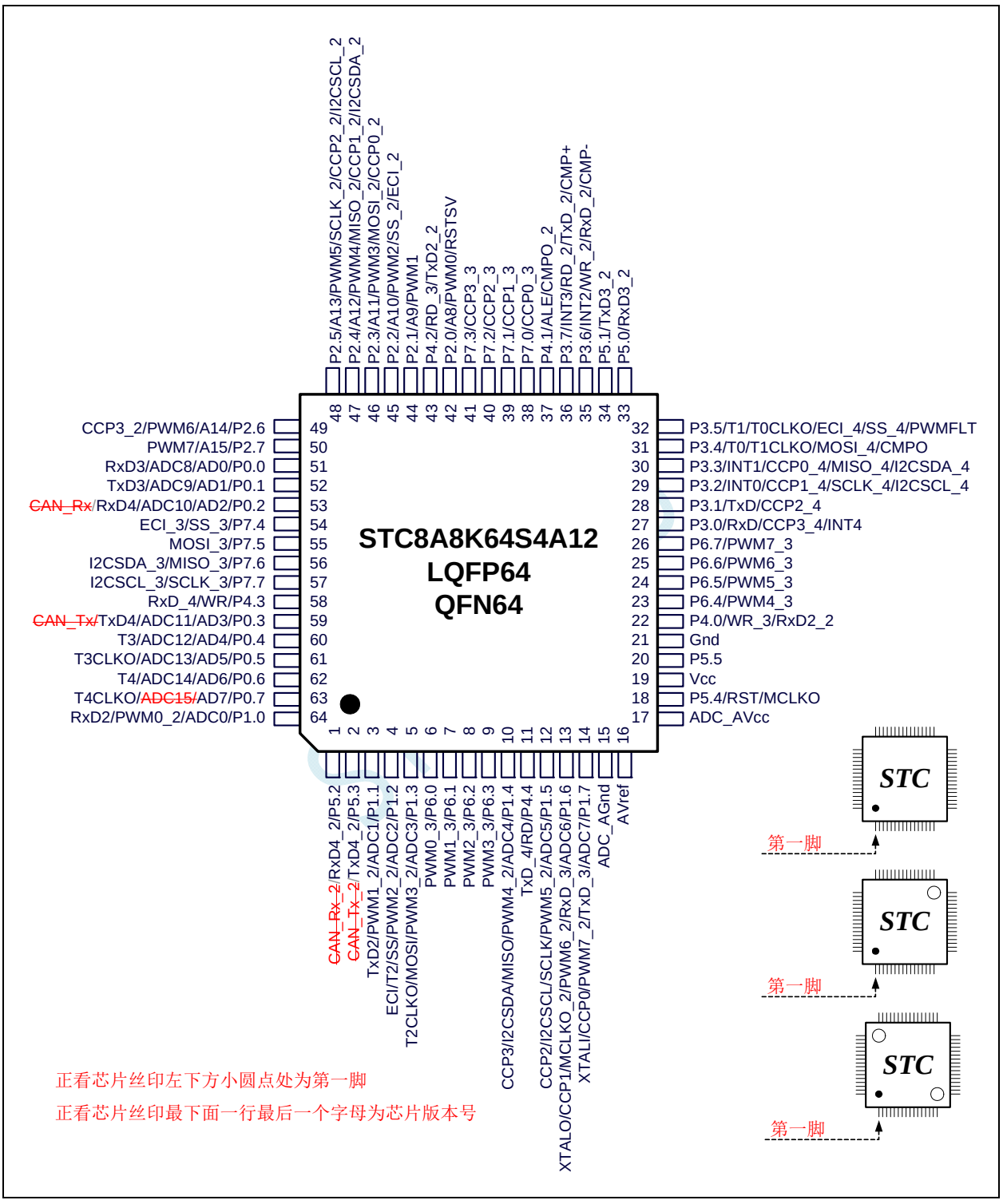
P7.0~P7.7

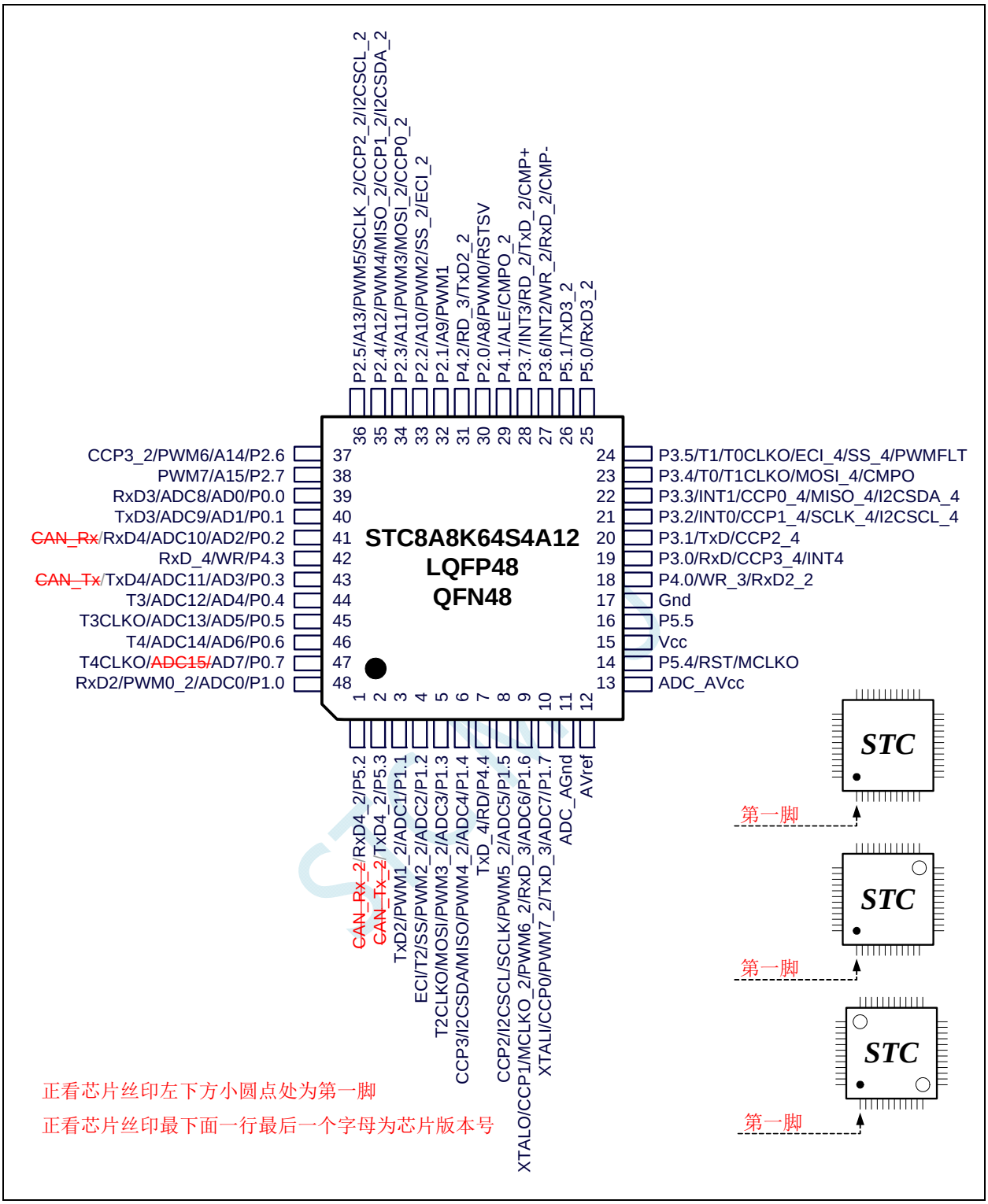
- ✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式

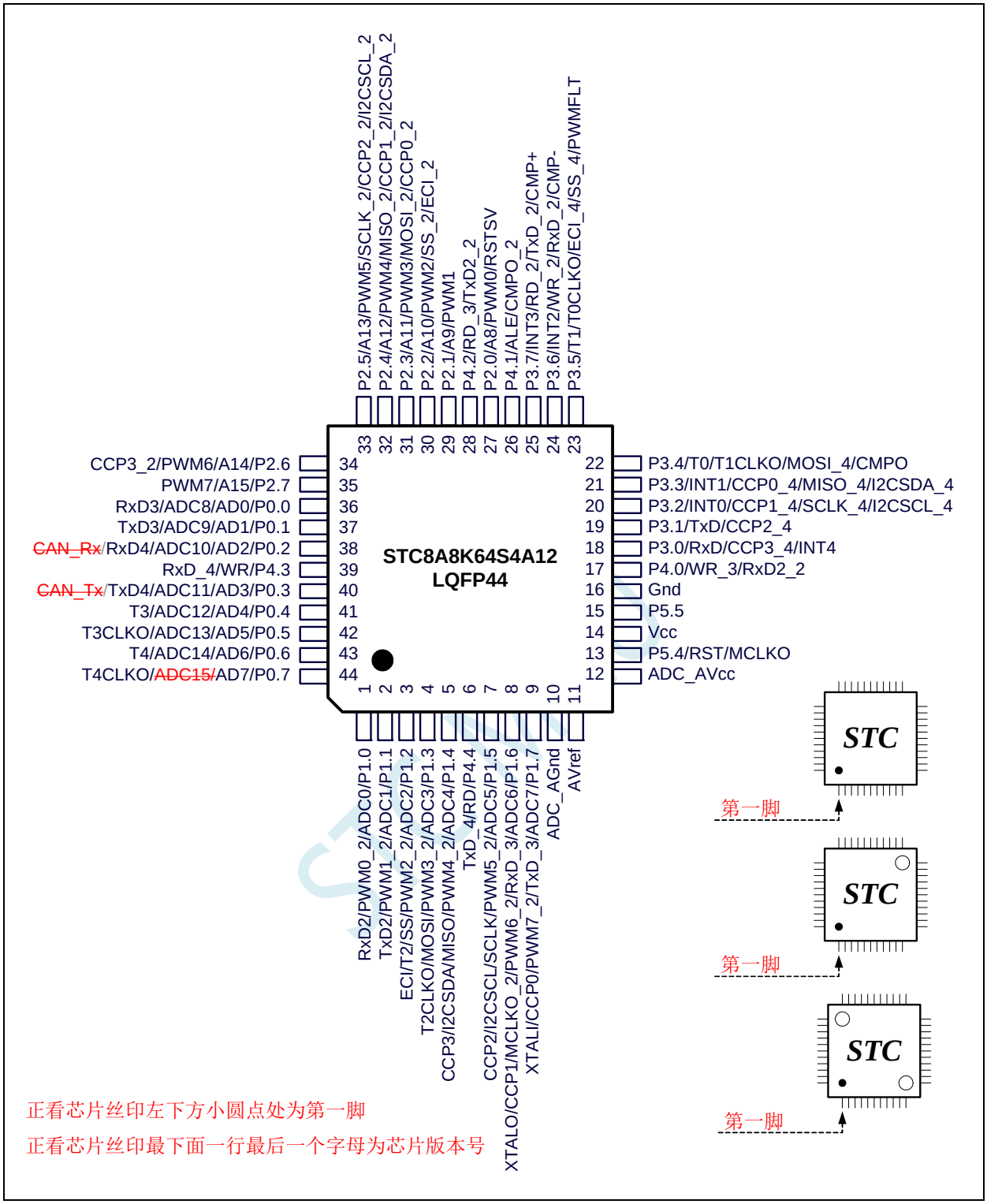
- ➢ **封装**
  - ✓ LQFP64S、LQFP48、LQFP44、~~PDIP40（暂未生产）~~

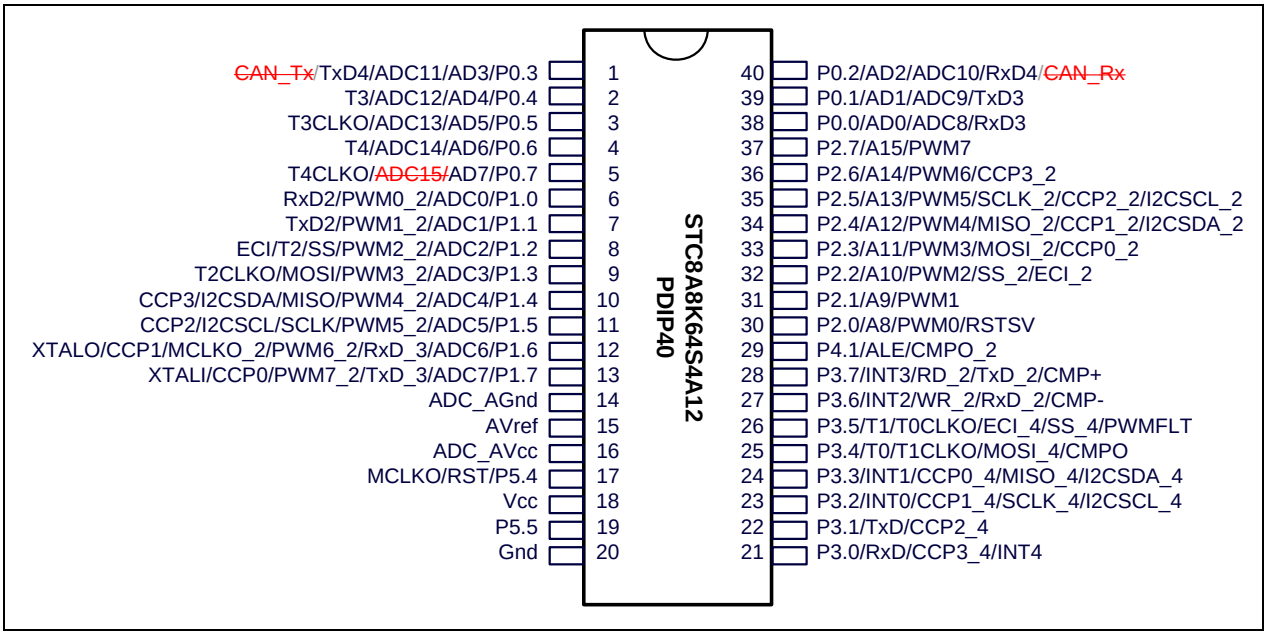
STC MCU

2.1.2 管脚图


STC8A8K64S4A12
LQFP64
QFN64
1 P2.5/A13/PWM5/SCLK_2/CCP2_2/I2CSCL_2
2 P2.4/A12/PWM4/MISO_2/CCP1_2/I2CSDA_2
3 P2.3/A11/PWM3/MOSI_2/CCP0_2
4 P2.2/A10/PWM2/SS_2/ECI_2
5 P2.1/A9/PWM1
6 P4.2/RD_3/TxD2_2
7 P2.0/A8/PWM0/RSTSV
8 P7.3/CCP3_3
9 P7.2/CCP2_3
10 P7.1/CCP1_3
11 P7.0/CCP0_3
12 P4.1/ALE/CMPO_2
13 P3.7/INT3/RD_2/TxD_2/CMP+
14 P3.6/INT2/WR_2/RxD_2/CMP-
15 P5.1/TxD3_2
16 P5.0/RxD3_2
17 ADC_AVcc
18 P5.4/RST/MCLKO
19 Vcc
20 P5.5
21 Gnd
22 P4.0/WR_3/RxD2_2
23 P6.4/PWM4_3
24 P6.5/PWM5_3
25 P6.6/PWM6_3
26 P6.7/PWM7_3
27 P3.0/RxD/CCP3_4/INT4
28 P3.1/TxD/CCP2_4
29 P3.2/INT0/CCP1_4/SCLK_4/I2CSCL_4
30 P3.3/INT1/CCP0_4/MISO_4/I2CSDA_4
31 P3.4/T0/T1CLKO/MOSI_4/CMPO
32 P3.5/T1/T0CLKO/ECI_4/SS_4/PWMFLT
33 AVref
34 ADC_AGnd
35 XTALI/CCP0/PWM7_2/TxD_3/ADC7/P1.7
36 XTALO/CCP1/MCLKO_2/PWM6_2/RxD_3/ADC6/P1.6
37 CCP2/I2CSCL/SCLK/PWM5_2/ADC5/P1.5
38 TxD_4/RD/P4.4
39 CCP3/I2CSDA/MISO/PWM4_2/ADC4/P1.4
40 PWM3_3/P6.3
41 PWM2_3/P6.2
42 PWM1_3/P6.1
43 PWM0_3/P6.0
44 T2CLKO/MOSI/PWM3_2/ADC3/P1.3
45 ECI/T2/SS/PWM2_2/ADC2/P1.2
46 TxD2/PWM1_2/ADC1/P1.1
47 ~~CAN_Tx_2~~/TxD4_2/P5.3
48 ~~CAN_Rx_2~~/RxD4_2/P5.2
49 CCP3_2/PWM6/A14/P2.6
50 PWM7/A15/P2.7
51 RxD3/ADC8/AD0/P0.0
52 TxD3/ADC9/AD1/P0.1
53 ~~CAN_Rx~~/RxD4/ADC10/AD2/P0.2
54 ECI_3/SS_3/P7.4
55 MOSI_3/P7.5
56 I2CSDA_3/MISO_3/P7.6
57 I2CSCL_3/SCLK_3/P7.7
58 RxD_4/WR/P4.3
59 ~~CAN_Tx~~/TxD4/ADC11/AD3/P0.3
60 T3/ADC12/AD4/P0.4
61 T3CLKO/ADC13/AD5/P0.5
62 T4/ADC14/AD6/P0.6
63 T4CLKO/~~ADC15~~/AD7/P0.7
64 RxD2/PWM0_2/ADC0/P1.0
正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号
STC
第一脚
STC
第一脚
STC
第一脚


P2.5/A13/PWM5/SCLK_2/CCP2_2/I2CSCL_2
P2.4/A12/PWM4/MISO_2/CCP1_2/I2CSDA_2
P2.3/A11/PWM3/MOSI_2/CCP0_2
P2.2/A10/PWM2/SS_2/ECI_2
P2.1/A9/PWM1
P4.2/RD_3/TxD2_2
P2.0/A8/PWM0/RSTSV
P4.1/ALE/CMPO_2
P3.7/INT3/RD_2/TxD_2/CMP+
P3.6/INT2/WR_2/RxD_2/CMP-
P5.1/TxD3_2
P5.0/RxD3_2
36 35 34 33 32 31 30 29 28 27 26 25
CCP3_2/PWM6/A14/P2.6 37
PWM7/A15/P2.7 38
RxD3/ADC8/AD0/P0.0 39
TxD3/ADC9/AD1/P0.1 40
~~CAN_Rx~~/RxD4/ADC10/AD2/P0.2 41
RxD_4/WR/P4.3 42
~~CAN_Tx~~/TxD4/ADC11/AD3/P0.3 43
T3/ADC12/AD4/P0.4 44
T3CLKO/ADC13/AD5/P0.5 45
T4/ADC14/AD6/P0.6 46
T4CLKO/~~ADC15/~~AD7/P0.7 47
RxD2/PWM0_2/ADC0/P1.0 48
STC8A8K64S4A12
LQFP48
QFN48
24 P3.5/T1/T0CLKO/ECI_4/SS_4/PWMFLT
23 P3.4/T0/T1CLKO/MOSI_4/CMPO
22 P3.3/INT1/CCP0_4/MISO_4/I2CSDA_4
21 P3.2/INT0/CCP1_4/SCLK_4/I2CSCL_4
20 P3.1/TxD/CCP2_4
19 P3.0/RxD/CCP3_4/INT4
18 P4.0/WR_3/RxD2_2
17 Gnd
16 P5.5
15 Vcc
14 P5.4/RST/MCLKO
13 ADC_AVcc
1 2 3 4 5 6 7 8 9 10 11 12
~~CAN_Rx_2~~/RxD4_2/P5.2
~~CAN_Tx_2~~/TxD4_2/P5.3
TxD2/PWM1_2/ADC1/P1.1
ECI/T2/SS/PWM2_2/ADC2/P1.2
T2CLKO/MOSI/PWM3_2/ADC3/P1.3
CCP3/I2CSDA/MISO/PWM4_2/ADC4/P1.4
TxD_4/RD/P4.4
CCP2/I2CSCL/SCLK/PWM5_2/ADC5/P1.5
XTALO/CCP1/MCLKO_2/PWM6_2/RxD_3/ADC6/P1.6
XTALI/CCP0/PWM7_2/TxD_3/ADC7/P1.7
ADC_AGnd
AVref
STC
第一脚
STC
第一脚
STC
第一脚
正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号


P2.5/A13/PWM5/SCLK_2/CCP2_2/I2CSCL_2
P2.4/A12/PWM4/MISO_2/CCP1_2/I2CSDA_2
P2.3/A11/PWM3/MOSI_2/CCP0_2
P2.2/A10/PWM2/SS_2/ECI_2
P2.1/A9/PWM1
P4.2/RD_3/TxD2_2
P2.0/A8/PWM0/RSTSV
P4.1/ALE/CMPO_2
P3.7/INT3/RD_2/TxD_2/CMP+
P3.6/INT2/WR_2/RxD_2/CMP-
P3.5/T1/T0CLKO/ECI_4/SS_4/PWMFLT
33 32 31 30 29 28 27 26 25 24 23
CCP3_2/PWM6/A14/P2.6 34
PWM7/A15/P2.7 35
RxD3/ADC8/AD0/P0.0 36
TxD3/ADC9/AD1/P0.1 37
~~CAN_Rx~~/RxD4/ADC10/AD2/P0.2 38
RxD_4/WR/P4.3 39
~~CAN_Tx~~/TxD4/ADC11/AD3/P0.3 40
T3/ADC12/AD4/P0.4 41
T3CLKO/ADC13/AD5/P0.5 42
T4/ADC14/AD6/P0.6 43
T4CLKO/~~ADC15~~/AD7/P0.7 44
STC8A8K64S4A12
LQFP44
22 P3.4/T0/T1CLKO/MOSI_4/CMPO
21 P3.3/INT1/CCP0_4/MISO_4/I2CSDA_4
20 P3.2/INT0/CCP1_4/SCLK_4/I2CSCL_4
19 P3.1/TxD/CCP2_4
18 P3.0/RxD/CCP3_4/INT4
17 P4.0/WR_3/RxD2_2
16 Gnd
15 P5.5
14 Vcc
13 P5.4/RST/MCLKO
12 ADC_AVcc
1 2 3 4 5 6 7 8 9 10 11
RxD2/PWM0_2/ADC0/P1.0
TxD2/PWM1_2/ADC1/P1.1
ECI/T2/SS/PWM2_2/ADC2/P1.2
T2CLKO/MOSI/PWM3_2/ADC3/P1.3
CCP3/I2CSDA/MISO/PWM4_2/ADC4/P1.4
TxD_4/RD/P4.4
CCP2/I2CSCL/SCLK/PWM5_2/ADC5/P1.5
XTALI/CCP0/PWM7_2/TxD_3/ADC7/P1.7
ADC_AGnd
AVref
XTALO/CCP1/MCLKO_2/PWM6_2/RxD_3/ADC6/P1.6
STC
第一脚
STC
第一脚
STC
第一脚
正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号


STC8A8K64S4A12
PDIP40
~~CAN_Tx~~/TxD4/ADC11/AD3/P0.3 1
T3/ADC12/AD4/P0.4 2
T3CLKO/ADC13/AD5/P0.5 3
T4/ADC14/AD6/P0.6 4
T4CLKO/~~ADC15~~/AD7/P0.7 5
RxD2/PWM0_2/ADC0/P1.0 6
TxD2/PWM1_2/ADC1/P1.1 7
ECI/T2/SS/PWM2_2/ADC2/P1.2 8
T2CLKO/MOSI/PWM3_2/ADC3/P1.3 9
CCP3/I2CSDA/MISO/PWM4_2/ADC4/P1.4 10
CCP2/I2CSCL/SCLK/PWM5_2/ADC5/P1.5 11
XTALO/CCP1/MCLKO_2/PWM6_2/RxD_3/ADC6/P1.6 12
XTALI/CCP0/PWM7_2/TxD_3/ADC7/P1.7 13
ADC_AGnd 14
AVref 15
ADC_AVcc 16
MCLKO/RST/P5.4 17
Vcc 18
P5.5 19
Gnd 20
40 P0.2/AD2/ADC10/RxD4/~~CAN_Rx~~
39 P0.1/AD1/ADC9/TxD3
38 P0.0/AD0/ADC8/RxD3
37 P2.7/A15/PWM7
36 P2.6/A14/PWM6/CCP3_2
35 P2.5/A13/PWM5/SCLK_2/CCP2_2/I2CSCL_2
34 P2.4/A12/PWM4/MISO_2/CCP1_2/I2CSDA_2
33 P2.3/A11/PWM3/MOSI_2/CCP0_2
32 P2.2/A10/PWM2/SS_2/ECI_2
31 P2.1/A9/PWM1
30 P2.0/A8/PWM0/RSTSV
29 P4.1/ALE/CMPO_2
28 P3.7/INT3/RD_2/TxD_2/CMP+
27 P3.6/INT2/WR_2/RxD_2/CMP-
26 P3.5/T1/T0CLKO/ECI_4/SS_4/PWMFLT
25 P3.4/T0/T1CLKO/MOSI_4/CMPO
24 P3.3/INT1/CCP0_4/MISO_4/I2CSDA_4
23 P3.2/INT0/CCP1_4/SCLK_4/I2CSCL_4
22 P3.1/TxD/CCP2_4
21 P3.0/RxD/CCP3_4/INT4

STC MCU

2.1.3 管脚说明

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
1	1			P5.2	I/O	标准 IO 口
				RxD4_2	I	串口 4 的接收脚
2	2			P5.3	I/O	标准 IO 口
				TxD4_2	O	串口 4 的发送脚
3	3	2	7	P1.1	I/O	标准 IO 口
				ADC1	I	ADC 模拟输入通道 1
				PWM1_2	O	增强 PWM 通道 1 输出脚
				TxD2	O	串口 2 的发送脚
4	4	3	8	P1.2	I/O	标准 IO 口
				ADC2	I	ADC 模拟输入通道 2
				PWM2_2	O	增强 PWM 通道 2 输出脚
				SS	I/O	SPI 从机选择
				T2	I	定时器 2 外部时钟输入
				ECI	I	PCA 的外部脉冲输入
5	5	4	9	P1.3	I/O	标准 IO 口
				ADC3	I	ADC 模拟输入通道 3
				PWM3_2	O	增强 PWM 通道 3 输出脚
				MOSI	I/O	SPI 主机输出从机输入
				T2CLKO	O	定时器 2 时钟分频输出
6				P6.0	I/O	标准 IO 口
				PWM0_3	O	增强 PWM 通道 0 输出脚
7				P6.1	I/O	标准 IO 口
				PWM1_3	O	增强 PWM 通道 1 输出脚
8				P6.2	I/O	标准 IO 口
				PWM2_3	O	增强 PWM 通道 2 输出脚
9				P6.3	I/O	标准 IO 口
				PWM3_3	O	增强 PWM 通道 3 输出脚

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
10	6	5	10	P1.4	I/O	标准 IO 口
				ADC4	I	ADC 模拟输入通道 4
				PWM4_2	O	增强 PWM 通道 4 输出脚
				MISO	I/O	SPI 主机输入从机输出
				SDA	I/O	I2C 接口的数据线
				CCP3	I/O	PCA 的捕获输入和脉冲输出
11	7	6		P4.4	I/O	标准 IO 口
				RD	O	外部总线的读信号线
				TxD_4	O	串口 1 的发送脚
12	8	7	11	P1.5	I/O	标准 IO 口
				ADC5	I	ADC 模拟输入通道 5
				PWM5_2	O	增强 PWM 通道 5 输出脚
				SCLK	I/O	SPI 的时钟脚
				SCL	I/O	I2C 的时钟线
				CCP2	I/O	PCA 的捕获输入和脉冲输出
13	9	8	12	P1.6	I/O	标准 IO 口
				ADC6	I	ADC 模拟输入通道 6
				RxD_3	I	串口 1 的接收脚
				PWM6_2	O	增强 PWM 通道 6 输出脚
				MCLKO_2	O	主时钟分频输出
				CCP1	I/O	PCA 的捕获输入和脉冲输出
				XTALO	O	外部晶振的输出脚
14	10	9	13	P1.7	I/O	标准 IO 口
				ADC7	I	ADC 模拟输入通道 7
				TxD_3	O	串口 1 的发送脚
				PWM7_2	O	增强 PWM 通道 7 输出脚
				CCP0	I/O	PCA 的捕获输入和脉冲输出
				XTALI	I	外部晶振/外部时钟的输入脚
15	11	10	14	ADC_AGnd	GND	ADC 地线
16	12	11	15	AVref	I	ADC 的参考电压脚
17	13	12	16	ADC_AVcc	VCC	ADC 电源脚
18	14	13	17	P5.4	I/O	标准 IO 口
				RST	I	复位引脚
				MCLKO	O	主时钟分频输出
19	15	14	18	Vcc	VCC	电源脚
20	16	15	19	P5.5	I/O	标准 IO 口
21	17	16	20	Gnd	GND	地线

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
22	18	17		P4.0	I/O	标准 IO 口
				WR_3	O	外部总线的写信号线
				RxD2_2	I	串口 2 的接收脚
23				P6.4	I/O	标准 IO 口
				PWM4_3	O	增强 PWM 通道 4 输出脚
24				P6.5	I/O	标准 IO 口
				PWM5_3	O	增强 PWM 通道 5 输出脚
25				P6.6	I/O	标准 IO 口
				PWM6_3	O	增强 PWM 通道 6 输出脚
26				P6.7	I/O	标准 IO 口
				PWM7_3	O	增强 PWM 通道 7 输出脚
27	19	18	21	P3.0	I/O	标准 IO 口
				RxD	I	串口 1 的接收脚
				CCP3_4	I/O	PCA 的捕获输入和脉冲输出
				INT4	I	外部中断 4
28	20	19	22	P3.1	I/O	标准 IO 口
				TxD	O	串口 1 的发送脚
				CCP2_4	I/O	PCA 的捕获输入和脉冲输出
29	21	20	23	P3.2	I/O	标准 IO 口
				INT0	I	外部中断 0
				CCP1_4	I/O	PCA 的捕获输入和脉冲输出
				SCLK_4	I/O	SPI 的时钟脚
				SCL_4	I/O	I2C 的时钟线
30	22	21	24	P3.3	I/O	标准 IO 口
				INT1	I	外部中断 1
				CCP0_4	I/O	PCA 的捕获输入和脉冲输出
				MISO_4	I/O	SPI 主机输入从机输出
				SDA_4	I/O	I2C 接口的数据线
31	23	22	25	P3.4	I/O	标准 IO 口
				T0	I	定时器 0 外部时钟输入
				T1CLKO	O	定时器 1 时钟分频输出
				MOSI_4	I/O	SPI 主机输出从机输入
				CMPO	O	比较器输出

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
32	24	23	26	P3.5	I/O	标准 IO 口
				T1	I	定时器 1 外部时钟输入
				T0CLKO	O	定时器 0 时钟分频输出
				ECI_4	I	PCA 的外部脉冲输入
				SS_4	I	SPI 的从机选择脚（主机为输出）
				PWMFLT	I	增强 PWM 的外部异常检测脚
33	25			P5.0	I/O	标准 IO 口
				RxD3_2	I	串口 3 的接收脚
34	26			P5.1	I/O	标准 IO 口
				TxD3_2	O	串口 3 的发送脚
35	27	24	27	P3.6	I/O	标准 IO 口
				INT2	I	外部中断 2
				WR_2	O	外部总线的写信号线
				RxD_2	I	串口 1 的接收脚
				CMP-	I	比较器负极输入
36	28	25	28	P3.7	I/O	标准 IO 口
				INT3	I	外部中断 3
				RD_2	O	外部总线的读信号线
				TxD_2	O	串口 1 的发送脚
				CMP+	I	比较器正极输入
37	29	26	29	P4.1	I/O	标准 IO 口
				ALE	O	地址锁存信号
				CMPO_2	O	比较器输出
38				P7.0	I/O	标准 IO 口
				CCP0_3	I/O	PCA 的捕获输入和脉冲输出
39				P7.1	I/O	标准 IO 口
				CCP1_3	I/O	PCA 的捕获输入和脉冲输出
40				P7.2	I/O	标准 IO 口
				CCP2_3	I/O	PCA 的捕获输入和脉冲输出
41				P7.3	I/O	标准 IO 口
				CCP3_3	I/O	PCA 的捕获输入和脉冲输出
42	30	27	30	P2.0	I/O	标准 IO 口
				A8	I	地址总线
				PWM0	O	增强 PWM 通道 0 输出脚
				RSTSV	-	端口的初始电平可 ISP 下载时配置
43	31	28		P4.2	I/O	标准 IO 口
				RD_3	O	外部总线的读信号线
				TxD2_2	O	串口 2 的发送脚

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
44	32	29	31	P2.1	I/O	标准 IO 口
				A9	I	地址总线
				PWM1	O	增强 PWM 通道 1 输出脚
45	33	30	32	P2.2	I/O	标准 IO 口
				A10	I	地址总线
				PWM2	O	增强 PWM 通道 2 输出脚
				SS_2	I	SPI 的从机选择脚（主机为输出）
				ECI_2	I	PCA 的外部脉冲输入
46	34	31	33	P2.3	I/O	标准 IO 口
				A11	I	地址总线
				PWM3	O	增强 PWM 通道 3 输出脚
				MOSI_2	I/O	SPI 主机输出从机输入
				CCP0_2	I/O	PCA 的捕获输入和脉冲输出
47	35	32	34	P2.4	I/O	标准 IO 口
				A12	I	地址总线
				PWM4	O	增强 PWM 通道 4 输出脚
				MISO_2	I/O	SPI 主机输入从机输出
				SDA_2	I/O	I2C 接口的数据线
				CCP1_2	I/O	PCA 的捕获输入和脉冲输出
48	36	33	35	P2.5	I/O	标准 IO 口
				A13	I	地址总线
				PWM5	O	增强 PWM 通道 5 输出脚
				SCLK_2	I/O	SPI 的时钟脚
				SCL_2	I/O	I2C 的时钟线
				CCP2_2	I/O	PCA 的捕获输入和脉冲输出
49	37	34	36	P2.6	I/O	标准 IO 口
				A14	I	地址总线
				PWM6	O	增强 PWM 通道 6 输出脚
				CCP3_2	I/O	PCA 的捕获输入和脉冲输出
50	38	35	37	P2.7	I/O	标准 IO 口
				A15	I	地址总线
				PWM7	O	增强 PWM 通道 7 输出脚
51	39	36	38	P0.0	I/O	标准 IO 口
				AD0	I	地址总线
				ADC8	I	ADC 模拟输入通道 8
				RxD3	I	串口 3 的接收脚

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
52	40	37	39	P0.1	I/O	标准 IO 口
				AD1	I	地址总线
				ADC9	I	ADC 模拟输入通道 9
				TxD3	O	串口 3 的发送脚
53	41	38	40	P0.2	I/O	标准 IO 口
				AD2	I	地址总线
				ADC10	I	ADC 模拟输入通道 10
				RxD4	I	串口 4 的接收脚
54				P7.4	I/O	标准 IO 口
				SS_3	I	SPI 的从机选择脚（主机为输出）
				ECI_3	I	PCA 的外部脉冲输入
55				P7.5	I/O	标准 IO 口
				MOSI_3	I/O	SPI 主机输出从机输入
56				P7.6	I/O	标准 IO 口
				MISO_3	I/O	SPI 主机输入从机输出
				SDA_3	I/O	I2C 接口的数据线
57				P7.7	I/O	标准 IO 口
				SCLK_3	I/O	SPI 的时钟脚
				SCL_3	I/O	I2C 的时钟线
58	42	39		P4.3	I/O	标准 IO 口
				WR	O	外部总线的写信号线
				RxD_4	I	串口 1 的接收脚
59	43	40	1	P0.3	I/O	标准 IO 口
				AD3	I	地址总线
				ADC11	I	ADC 模拟输入通道 11
				TxD4	O	串口 4 的发送脚
60	44	41	2	P0.4	I/O	标准 IO 口
				AD4	I	地址总线
				ADC12	I	ADC 模拟输入通道 12
				T3	I	定时器 3 外部时钟输入

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
61	45	42	3	P0.5	I/O	标准 IO 口
				AD5	I	地址总线
				ADC13	I	ADC 模拟输入通道 13
				T3CLKO	O	定时器 3 时钟分频输出
62	46	43	4	P0.6	I/O	标准 IO 口
				AD6	I	地址总线
				ADC14	I	ADC 模拟输入通道 14
				T4	I	定时器 4 外部时钟输入
63	47	44	5	P0.7	I/O	标准 IO 口
				AD7	I	地址总线
				T4CLKO	O	定时器 4 时钟分频输出
64	48	1	6	P1.0	I/O	标准 IO 口
				ADC0	I	ADC 模拟输入通道 0
				PWM0_2	O	增强 PWM 通道 0 输出脚
				RxD2	I	串口 2 的接收脚

STC MCU

2.2 STC8A4K64S2A12-64Pin系列

2.2.1 特性及价格

➢ 选型价格（不需要外部晶振、不需要外部复位，12 位 ADC，15 通道）

单片机型号	工作电压（V）	Flash 程序存储器 10 万次 字节	大容量扩展 SRAM 字节	强大的双 DPTR 可增可减	EEPROM 10 万次 字节	I/O 口最多数量	串口并可掉电唤醒	SPI	I²C	定时器/计数器（T0-T4 外部管脚也可掉电唤醒）	16 位高级 PWM 定时器 互补对称死区	15 位增强型 PWM（带死区控制）	PCA/CCP/PWM（可当外部中断并可掉电唤醒）	掉电唤醒专用定时器	15 路高速 ADC(8 路 PWM 可当 8 路 D/A 使用)	比较器（可当 1 路 A/D，可作外部掉电检测）	内部低压检测中断并可掉电唤醒	看门狗 复位定时器	内部高可靠复位（可选复位门槛电压）	内部高精准时钟（24MHz 可调）	可对外输出时钟及复位	程序加密后传输（防拦截）	可设置下次更新程序需口令	支持 RS485 下载	支持 USB 直接下载	本身就可在线仿真	封装						2017 年新品供货信息
																											LQFP64S	QFN64 <8x8mm>	LQFP48	QFN48 <6x6mm>	LQFP44	PDIP40	
STC8A4K16S2A12	2.0-5.5	16K	4K	2	48K	59	2	有	有	5	-	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	¥3.1		¥2.9		¥2.9		大量供货
STC8A4K32S2A12	2.0-5.5	32K	4K	2	32K	59	2	有	有	5	-	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	¥3.3		¥3.0		¥3.0		
STC8A4K60S2A12	2.0-5.5	60K	4K	2	4K	59	2	有	有	5	-	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	¥3.6		¥3.2		¥3.2		
STC8A4K64S2A12	2.0-5.5	64K	4K	2	IAP	59	2	有	有	5	-	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	¥3.6		¥3.2		¥3.2		

注：以上的单价为 10K 及以上订货量的价格，量小则每片需增加 0.1 元人民币。当订货的总额达到或高于 3000 元时，可免运费发货，否则需要由客户承担运费。零售 10 片起售。

➢ **内核**
- ✓ 超高速 8051 内核（1T），比传统 8051 约快 12 倍以上
- ✓ 指令代码完全兼容传统 8051
- ✓ 20 个中断源，4 级中断优先级
- ✓ 支持在线仿真

➢ **工作电压**
- ✓ 2.0V～5.5V
- ✓ 内建 LDO

➢ **工作温度**
- ✓ -40℃～85℃（超温度范围应用请参考电器特性章节说明）

➢ **Flash 存储器**
- ✓ 最大 64K 字节 FLASH 程序存储器（ROM），用于存储用户代码
- ✓ 支持用户配置 EEPROM 大小，512 字节单页擦除，擦写次数可达 10 万次以上
- ✓ 支持在系统编程方式（ISP）更新用户应用程序，无需专用编程器
- ✓ 支持单芯片仿真，无需专用仿真器，理论断点个数无限制

➢ **SRAM**
- ✓ 128 字节内部直接访问 RAM（DATA）
- ✓ 128 字节内部间接访问 RAM（IDATA）
- ✓ 4096 字节内部扩展 RAM（内部 XDATA）
- ✓ 外部最大可扩展 64K 字节 RAM（外部 XDATA）

- **时钟控制**
  - ✓ 内部 24MHz 高精度 IRC（ISP 编程时可进行上下调整）
    - ⌖ 误差±0.3%（常温下 25℃）
    - ⌖ -1.8%～+0.8%温漂（全温度范围，-40℃～85℃）
    - ⌖ -1.0%～+0.5%温漂（温度范围，-20℃～65℃）
  - ✓ 内部 32KHz 低速 IRC（误差较大）
  - ✓ 外部晶振（4MHz～33MHz）和外部时钟

    用户可自由选择上面的 3 种时钟源

- **复位**
  - ✓ 硬件复位
    - ⌖ 上电复位
    - ⌖ 复位脚复位（高电平复位），出厂时 P5.4 默认为 IO 口，ISP 下载时可将 P5.4 管脚设置为复位脚
    - ⌖ 看门狗溢出复位
    - ⌖ 低压检测复位，提供 4 级低压检测电压：2.2V、2.4V、2.7V、3.0V
  - ✓ 软件复位
    - ⌖ 软件方式写复位触发寄存器

- **中断**
  - ✓ 提供 20 个中断源：INT0、INT1、INT2、INT3、INT4、定时器 0、定时器 1、定时器 2、定时器 3、定时器 4、串口 1、串口 2、ADC 模数转换、LVD 低压检测、SPI、I^2C、比较器、PCA/CCP/PWM、增强型 PWM、增强型 PWM 异常检测
  - ✓ 提供 4 级中断优先级

- **数字外设**
  - ✓ 5 个 16 位定时器：定时器 0、定时器 1、定时器 2、定时器 3、定时器 4，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式
  - ✓ 2 个高速串口：串口 1、串口 2 波特率时钟源最快可为 FOSC/4
  - ✓ 4 组 16 位 PCA 模块：CCP0、CCP1、CCP2、CCP3，可用于捕获、高速脉冲输出，及 6/7/8/10 位的 PWM 输出
  - ✓ 8 组 15 位增强型 PWM，可实现带死区的控制信号，并支持外部异常检测功能，另外还有 4 组传统的 PCA/CCP/PWM 可作 PWM
  - ✓ SPI：支持主机模式和从机模式以及主机/从机自动切换
  - ✓ I^2C：支持主机模式和从机模式

- **模拟外设**
  - ✓ 超高速 ADC，支持 **12 位精度** 15 通道（通道 0～通道 14）的模数转换，**速度最快能达到 800K（每秒进行 80 万次 ADC 转换）**
  - ✓ ADC 的通道 15 用于测试内部 1.344V 参考信号源（芯片在出厂时，内部参考信号源已调整为 1.344V）
  - ✓ 比较器，一组比较器（比较器的正端可选择 CMP+端口和所有的 ADC 输入端口，所以比较器可当作多路比较器进行分时复用）
  - ✓ DAC：4 路 PCA/CCP/PWM 可当 4 路 DAC 使用

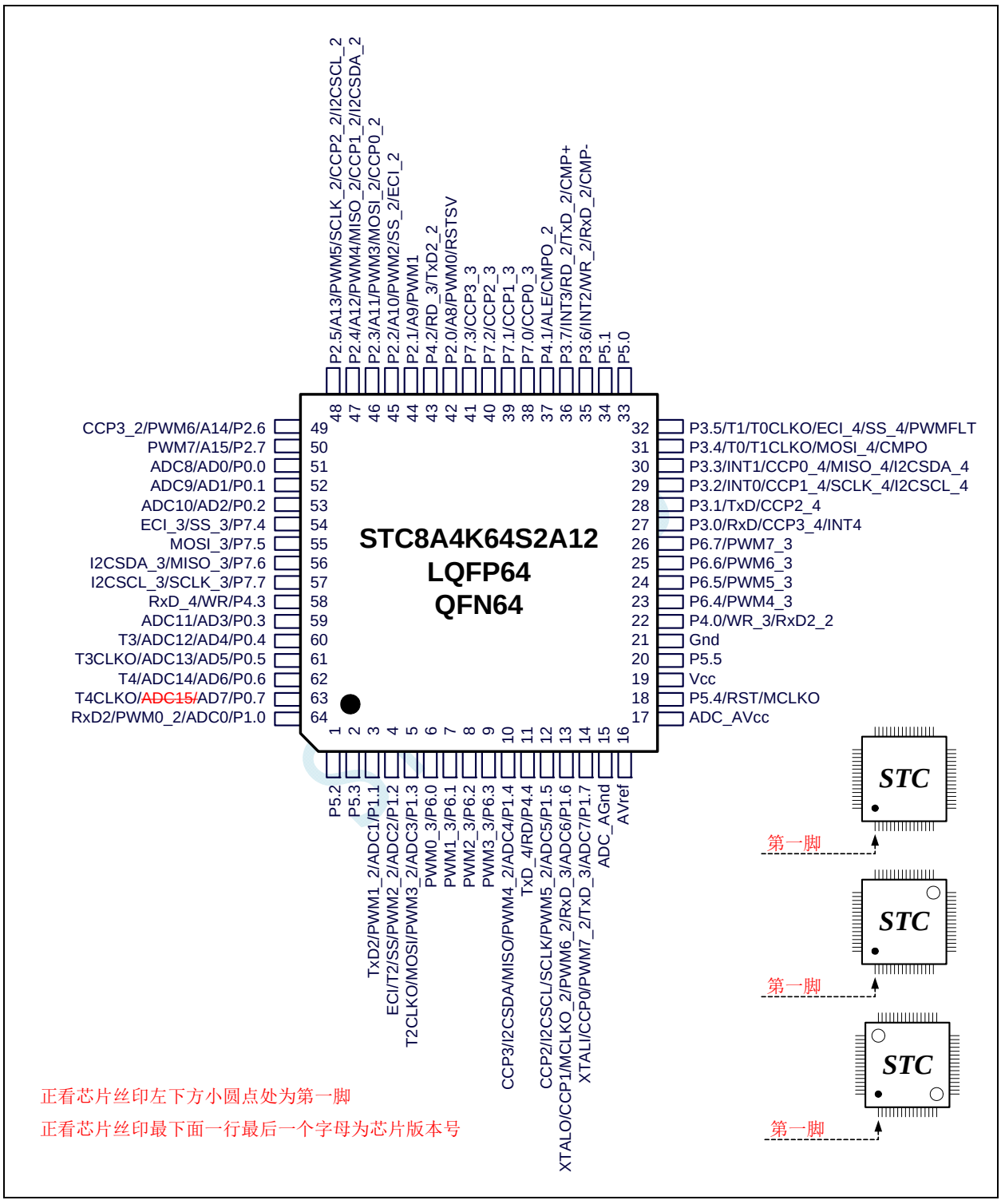
- **GPIO**
  - ✓ 最多可达 59 个 GPIO：P0.0~P0.7、P1.0~P1.7、P2.0~P2.7、P3.0~P3.7、P4.0~P4.4、P5.0~P5.5、P6.0~P6.7、P7.0~P7.7
  - ✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式

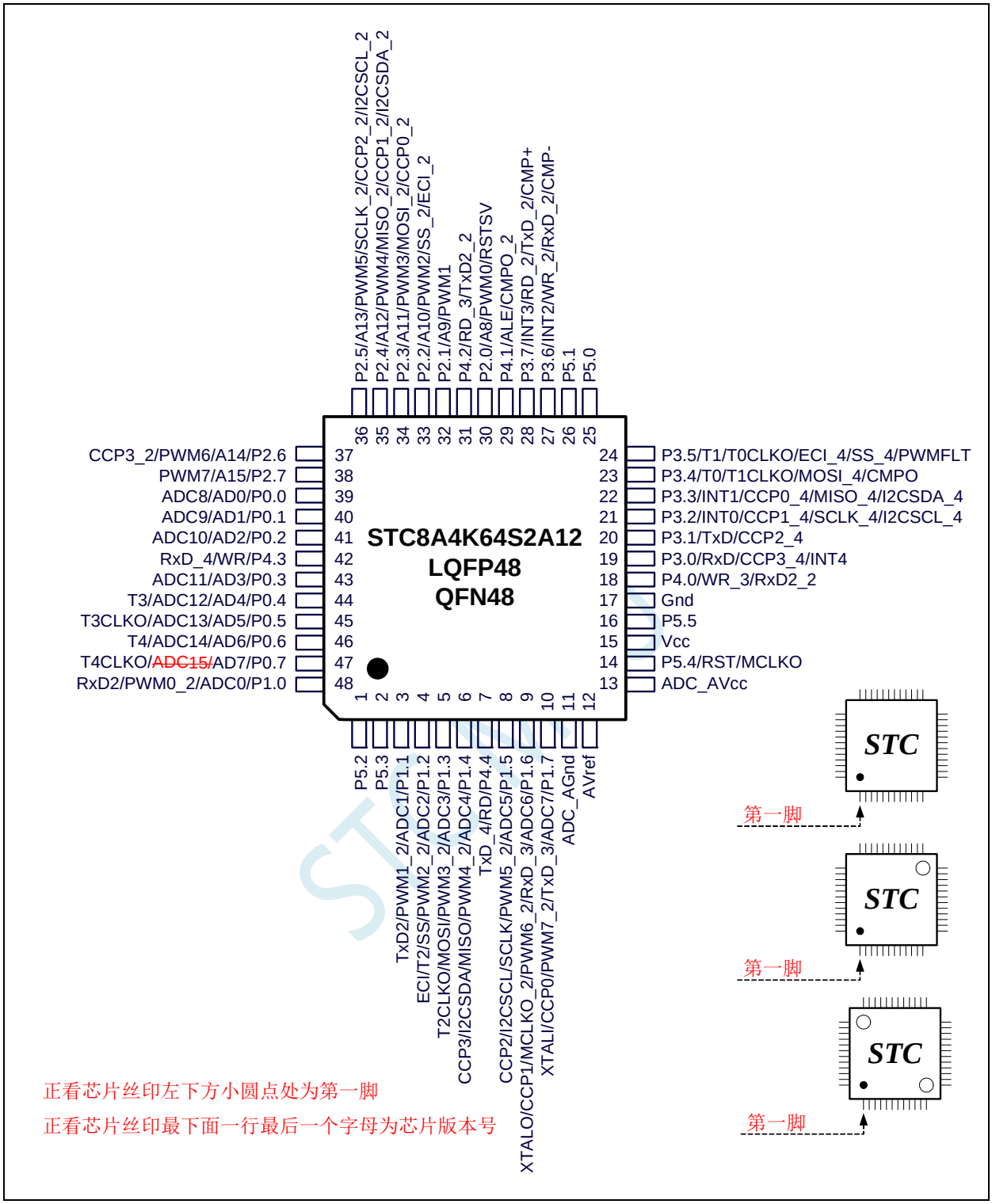
- **封装**

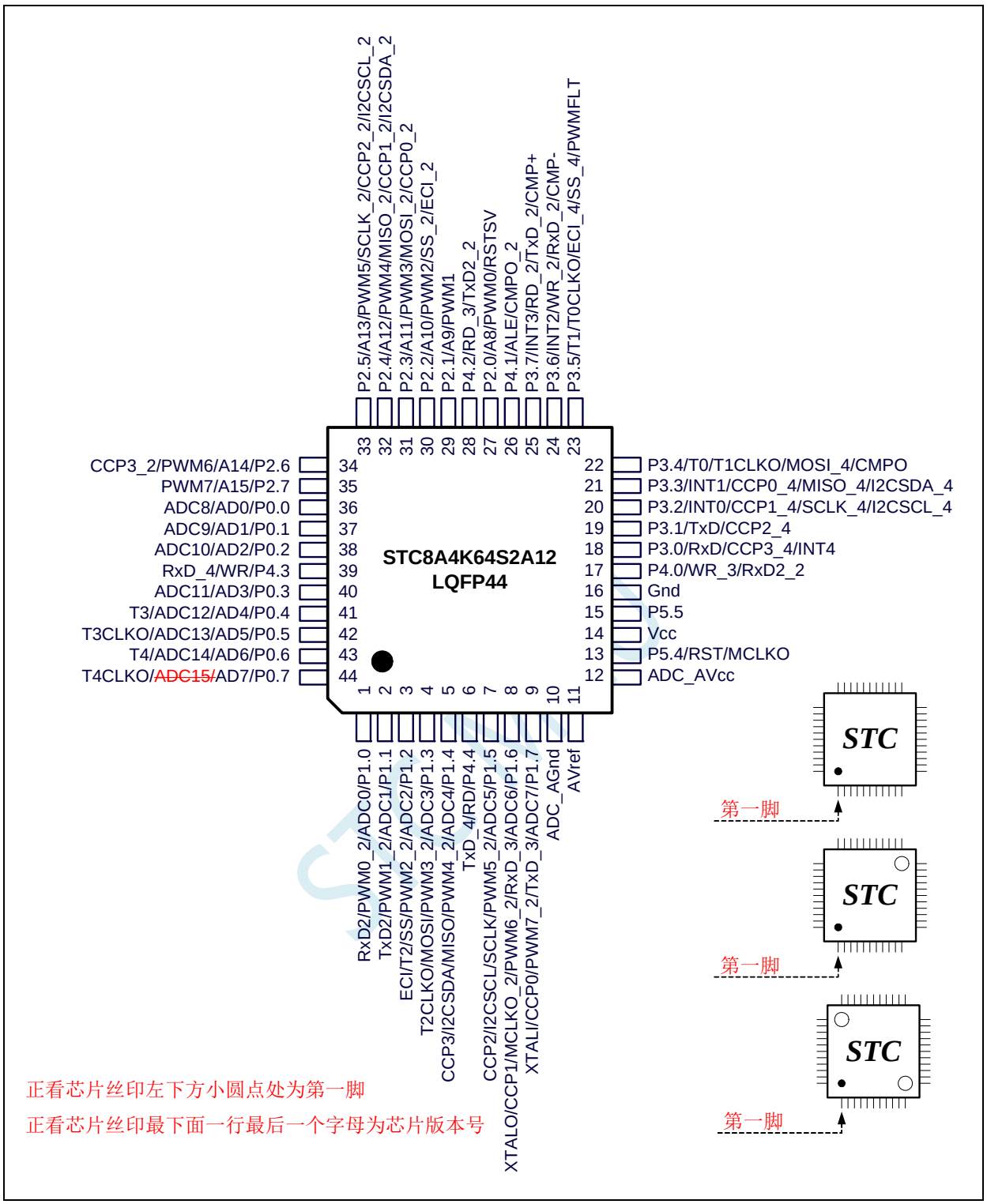
- ✓ LQFP64S、LQFP48、LQFP44、~~PDIP40（暂未生产）~~

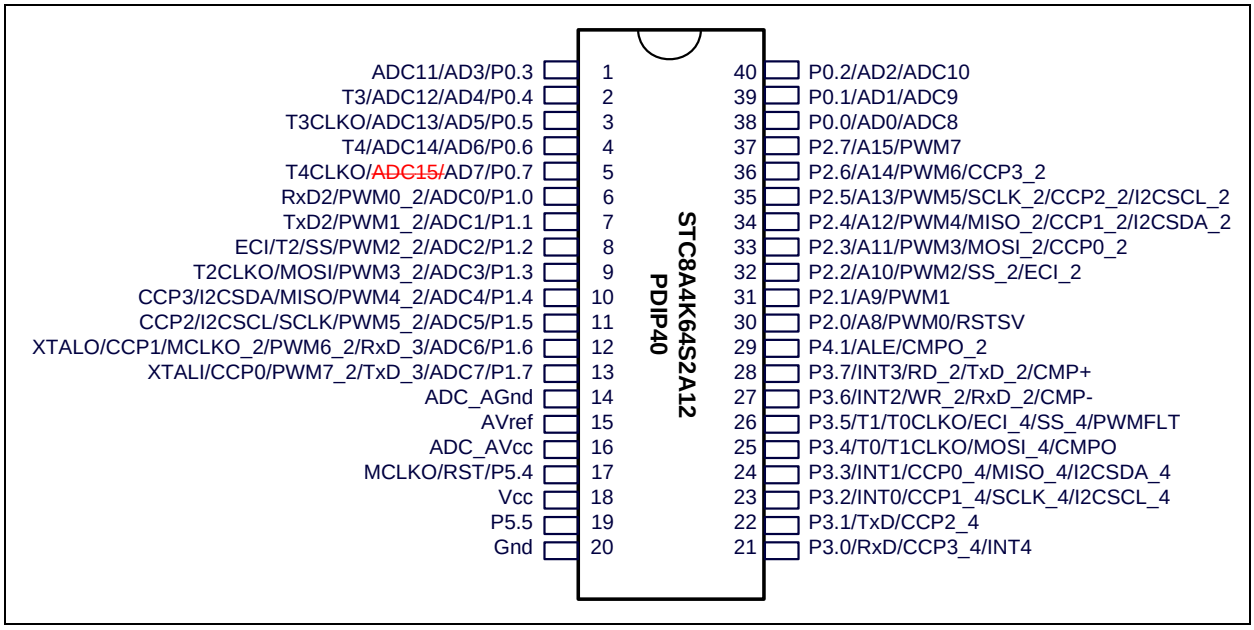
STC MCU

2.2.2 管脚图


STC8A4K64S2A12
LQFP64
QFN64
1 P5.2
2 P5.3
3 TxD2/PWM1_2/ADC1/P1.1
4 ECI/T2/SS/PWM2_2/ADC2/P1.2
5 T2CLKO/MOSI/PWM3_2/ADC3/P1.3
6 PWM0_3/P6.0
7 PWM1_3/P6.1
8 PWM2_3/P6.2
9 PWM3_3/P6.3
10 CCP3/I2CSDA/MISO/PWM4_2/ADC4/P1.4
11 TxD_4/RD/P4.4
12 CCP2/I2CSCL/SCLK/PWM5_2/ADC5/P1.5
13 XTALO/CCP1/MCLKO_2/PWM6_2/RxD_3/ADC6/P1.6
14 XTALI/CCP0/PWM7_2/TxD_3/ADC7/P1.7
15 ADC_AGnd
16 AVref
17 ADC_AVcc
18 P5.4/RST/MCLKO
19 Vcc
20 P5.5
21 Gnd
22 P4.0/WR_3/RxD2_2
23 P6.4/PWM4_3
24 P6.5/PWM5_3
25 P6.6/PWM6_3
26 P6.7/PWM7_3
27 P3.0/RxD/CCP3_4/INT4
28 P3.1/TxD/CCP2_4
29 P3.2/INT0/CCP1_4/SCLK_4/I2CSCL_4
30 P3.3/INT1/CCP0_4/MISO_4/I2CSDA_4
31 P3.4/T0/T1CLKO/MOSI_4/CMPO
32 P3.5/T1/T0CLKO/ECI_4/SS_4/PWMFLT
33 P5.0
34 P5.1
35 P3.6/INT2/WR_2/RxD_2/CMP-
36 P3.7/INT3/RD_2/TxD_2/CMP+
37 P4.1/ALE/CMPO_2
38 P7.0/CCP0_3
39 P7.1/CCP1_3
40 P7.2/CCP2_3
41 P7.3/CCP3_3
42 P2.0/A8/PWM0/RSTSV
43 P4.2/RD_3/TxD2_2
44 P2.1/A9/PWM1
45 P2.2/A10/PWM2/SS_2/ECI_2
46 P2.3/A11/PWM3/MOSI_2/CCP0_2
47 P2.4/A12/PWM4/MISO_2/CCP1_2/I2CSDA_2
48 P2.5/A13/PWM5/SCLK_2/CCP2_2/I2CSCL_2
49 CCP3_2/PWM6/A14/P2.6
50 PWM7/A15/P2.7
51 ADC8/AD0/P0.0
52 ADC9/AD1/P0.1
53 ADC10/AD2/P0.2
54 ECI_3/SS_3/P7.4
55 MOSI_3/P7.5
56 I2CSDA_3/MISO_3/P7.6
57 I2CSCL_3/SCLK_3/P7.7
58 RxD_4/WR/P4.3
59 ADC11/AD3/P0.3
60 T3/ADC12/AD4/P0.4
61 T3CLKO/ADC13/AD5/P0.5
62 T4/ADC14/AD6/P0.6
63 T4CLKO/~~ADC15/~~AD7/P0.7
64 RxD2/PWM0_2/ADC0/P1.0
STC
第一脚
STC
第一脚
STC
第一脚
正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号


STC8A4K64S2A12
LQFP48
QFN48
1 P5.2
2 P5.3
3 TxD2/PWM1_2/ADC1/P1.1
4 ECI/T2SS/PWM2_2/ADC2/P1.2
5 T2CLKO/MOSI/PWM3_2/ADC3/P1.3
6 CCP3/I2CSDA/MISO/PWM4_2/ADC4/P1.4
7 TxD_4/RD/P4.4
8 CCP2/I2CSCL/SCLK/PWM5_2/ADC5/P1.5
9 XTALO/CCP1/MCLKO_2/PWM6_2/RxD_3/ADC6/P1.6
10 XTALI/CCP0/PWM7_2/TxD_3/ADC7/P1.7
11 ADC_AGnd
12 AVref
13 ADC_AVcc
14 P5.4/RST/MCLKO
15 Vcc
16 P5.5
17 Gnd
18 P4.0/WR_3/RxD2_2
19 P3.0/RxD/CCP3_4/INT4
20 P3.1/TxD/CCP2_4
21 P3.2/INT0/CCP1_4/SCLK_4/I2CSCL_4
22 P3.3/INT1/CCP0_4/MISO_4/I2CSDA_4
23 P3.4/T0/T1CLKO/MOSI_4/CMPO
24 P3.5/T1/T0CLKO/ECI_4/SS_4/PWMFLT
25 P5.0
26 P5.1
27 P3.6/INT2/WR_2/RxD_2/CMP-
28 P3.7/INT3/RD_2/TxD_2/CMP+
29 P4.1/ALE/CMPO_2
30 P2.0/A8/PWM0/RSTSV
31 P4.2/RD_3/TxD2_2
32 P2.1/A9/PWM1
33 P2.2/A10/PWM2/SS_2/ECI_2
34 P2.3/A11/PWM3/MOSI_2/CCP0_2
35 P2.4/A12/PWM4/MISO_2/CCP1_2/I2CSDA_2
36 P2.5/A13/PWM5/SCLK_2/CCP2_2/I2CSCL_2
37 CCP3_2/PWM6/A14/P2.6
38 PWM7/A15/P2.7
39 ADC8/AD0/P0.0
40 ADC9/AD1/P0.1
41 ADC10/AD2/P0.2
42 RxD_4/WR/P4.3
43 ADC11/AD3/P0.3
44 T3/ADC12/AD4/P0.4
45 T3CLKO/ADC13/AD5/P0.5
46 T4/ADC14/AD6/P0.6
47 T4CLKO/~~ADC15/~~AD7/P0.7
48 RxD2/PWM0_2/ADC0/P1.0
STC
第一脚
STC
第一脚
STC
第一脚
正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号


P2.5/A13/PWM5/SCLK_2/CCP2_2/I2CSCL_2
P2.4/A12/PWM4/MISO_2/CCP1_2/I2CSDA_2
P2.3/A11/PWM3/MOSI_2/CCP0_2
P2.2/A10/PWM2/SS_2/ECI_2
P2.1/A9/PWM1
P4.2/RD_3/TxD2_2
P2.0/A8/PWM0/RSTSV
P4.1/ALE/CMPO_2
P3.7/INT3/RD_2/TxD_2/CMP+
P3.6/INT2/WR_2/RxD_2/CMP-
P3.5/T1/T0CLKO/ECI_4/SS_4/PWMFLT
33 32 31 30 29 28 27 26 25 24 23
CCP3_2/PWM6/A14/P2.6 34
PWM7/A15/P2.7 35
ADC8/AD0/P0.0 36
ADC9/AD1/P0.1 37
ADC10/AD2/P0.2 38
RxD_4/WR/P4.3 39
ADC11/AD3/P0.3 40
T3/ADC12/AD4/P0.4 41
T3CLKO/ADC13/AD5/P0.5 42
T4/ADC14/AD6/P0.6 43
T4CLKO/ADC15/AD7/P0.7 44
22 P3.4/T0/T1CLKO/MOSI_4/CMPO
21 P3.3/INT1/CCP0_4/MISO_4/I2CSDA_4
20 P3.2/INT0/CCP1_4/SCLK_4/I2CSCL_4
19 P3.1/TxD/CCP2_4
18 P3.0/RxD/CCP3_4/INT4
17 P4.0/WR_3/RxD2_2
16 Gnd
15 P5.5
14 Vcc
13 P5.4/RST/MCLKO
12 ADC_AVcc
STC8A4K64S2A12
LQFP44
1 2 3 4 5 6 7 8 9 10 11
RxD2/PWM0_2/ADC0/P1.0
TxD2/PWM1_2/ADC1/P1.1
ECI/T2SS/PWM2_2/ADC2/P1.2
T2CLKO/MOSI/PWM3_2/ADC3/P1.3
CCP3/I2CSDA/MISO/PWM4_2/ADC4/P1.4
TxD_4/RD/P4.4
CCP2/I2CSCL/SCLK/PWM5_2/ADC5/P1.5
XTALO/CCP1/MCLKO_2/PWM6_2/RxD_3/ADC6/P1.6
XTALI/CCP0/PWM7_2/TxD_3/ADC7/P1.7
ADC_AGnd
AVref
STC
第一脚
STC
第一脚
STC
第一脚
正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号


STC8A4K64S2A12
PDIP40
1 ADC11/AD3/P0.3
2 T3/ADC12/AD4/P0.4
3 T3CLKO/ADC13/AD5/P0.5
4 T4/ADC14/AD6/P0.6
5 T4CLKO/~~ADC15/~~AD7/P0.7
6 RxD2/PWM0_2/ADC0/P1.0
7 TxD2/PWM1_2/ADC1/P1.1
8 ECI/T2/SS/PWM2_2/ADC2/P1.2
9 T2CLKO/MOSI/PWM3_2/ADC3/P1.3
10 CCP3/I2CSDA/MISO/PWM4_2/ADC4/P1.4
11 CCP2/I2CSCL/SCLK/PWM5_2/ADC5/P1.5
12 XTALO/CCP1/MCLKO_2/PWM6_2/RxD_3/ADC6/P1.6
13 XTALI/CCP0/PWM7_2/TxD_3/ADC7/P1.7
14 ADC_AGnd
15 AVref
16 ADC_AVcc
17 MCLKO/RST/P5.4
18 Vcc
19 P5.5
20 Gnd
40 P0.2/AD2/ADC10
39 P0.1/AD1/ADC9
38 P0.0/AD0/ADC8
37 P2.7/A15/PWM7
36 P2.6/A14/PWM6/CCP3_2
35 P2.5/A13/PWM5/SCLK_2/CCP2_2/I2CSCL_2
34 P2.4/A12/PWM4/MISO_2/CCP1_2/I2CSDA_2
33 P2.3/A11/PWM3/MOSI_2/CCP0_2
32 P2.2/A10/PWM2/SS_2/ECI_2
31 P2.1/A9/PWM1
30 P2.0/A8/PWM0/RSTSV
29 P4.1/ALE/CMPO_2
28 P3.7/INT3/RD_2/TxD_2/CMP+
27 P3.6/INT2/WR_2/RxD_2/CMP-
26 P3.5/T1/T0CLKO/ECI_4/SS_4/PWMFLT
25 P3.4/T0/T1CLKO/MOSI_4/CMPO
24 P3.3/INT1/CCP0_4/MISO_4/I2CSDA_4
23 P3.2/INT0/CCP1_4/SCLK_4/I2CSCL_4
22 P3.1/TxD/CCP2_4
21 P3.0/RxD/CCP3_4/INT4

STC MCU

2.2.3 管脚说明

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
1	1			P5.2	I/O	标准 IO 口
2	2			P5.3	I/O	标准 IO 口
3	3	2	7	P1.1	I/O	标准 IO 口
				ADC1	I	ADC 模拟输入通道 1
				PWM1_2	O	增强 PWM 通道 1 输出脚
				TxD2	O	串口 2 的发送脚
4	4	3	8	P1.2	I/O	标准 IO 口
				ADC2	I	ADC 模拟输入通道 2
				PWM2_2	O	增强 PWM 通道 2 输出脚
				SS	I/O	SPI 从机选择
				T2	I	定时器 2 外部时钟输入
				ECI	I	PCA 的外部脉冲输入
5	5	4	9	P1.3	I/O	标准 IO 口
				ADC3	I	ADC 模拟输入通道 3
				PWM3_2	O	增强 PWM 通道 3 输出脚
				MOSI	I/O	SPI 主机输出从机输入
				T2CLKO	O	定时器 2 时钟分频输出
6				P6.0	I/O	标准 IO 口
				PWM0_3	O	增强 PWM 通道 0 输出脚
7				P6.1	I/O	标准 IO 口
				PWM1_3	O	增强 PWM 通道 1 输出脚
8				P6.2	I/O	标准 IO 口
				PWM2_3	O	增强 PWM 通道 2 输出脚
9				P6.3	I/O	标准 IO 口
				PWM3_3	O	增强 PWM 通道 3 输出脚

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
10	6	5	10	P1.4	I/O	标准 IO 口
				ADC4	I	ADC 模拟输入通道 4
				PWM4_2	O	增强 PWM 通道 4 输出脚
				MISO	I/O	SPI 主机输入从机输出
				SDA	I/O	I2C 接口的数据线
				CCP3	I/O	PCA 的捕获输入和脉冲输出
11	7	6		P4.4	I/O	标准 IO 口
				RD	O	外部总线的读信号线
				TxD_4	O	串口 1 的发送脚
12	8	7	11	P1.5	I/O	标准 IO 口
				ADC5	I	ADC 模拟输入通道 5
				PWM5_2	O	增强 PWM 通道 5 输出脚
				SCLK	I/O	SPI 的时钟脚
				SCL	I/O	I2C 的时钟线
				CCP2	I/O	PCA 的捕获输入和脉冲输出
13	9	8	12	P1.6	I/O	标准 IO 口
				ADC6	I	ADC 模拟输入通道 6
				RxD_3	I	串口 1 的接收脚
				PWM6_2	O	增强 PWM 通道 6 输出脚
				MCLKO_2	O	主时钟分频输出
				CCP1	I/O	PCA 的捕获输入和脉冲输出
				XTALO	O	外部晶振的输出脚
14	10	9	13	P1.7	I/O	标准 IO 口
				ADC7	I	ADC 模拟输入通道 7
				TxD_3	O	串口 1 的发送脚
				PWM7_2	O	增强 PWM 通道 7 输出脚
				CCP0	I/O	PCA 的捕获输入和脉冲输出
				XTALI	I	外部晶振/外部时钟的输入脚
15	11	10	14	ADC_AGnd	GND	ADC 地线
16	12	11	15	AVref	I	ADC 的参考电压脚
17	13	12	16	ADC_AVcc	VCC	ADC 电源脚
18	14	13	17	P5.4	I/O	标准 IO 口
				RST	I	复位引脚
				MCLKO	O	主时钟分频输出
19	15	14	18	Vcc	VCC	电源脚
20	16	15	19	P5.5	I/O	标准 IO 口
21	17	16	20	Gnd	GND	地线

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
22	18	17		P4.0	I/O	标准 IO 口
				WR_3	O	外部总线的写信号线
				RxD2_2	I	串口 2 的接收脚
23				P6.4	I/O	标准 IO 口
				PWM4_3	O	增强 PWM 通道 4 输出脚
24				P6.5	I/O	标准 IO 口
				PWM5_3	O	增强 PWM 通道 5 输出脚
25				P6.6	I/O	标准 IO 口
				PWM6_3	O	增强 PWM 通道 6 输出脚
26				P6.7	I/O	标准 IO 口
				PWM7_3	O	增强 PWM 通道 7 输出脚
27	19	18	21	P3.0	I/O	标准 IO 口
				RxD	I	串口 1 的接收脚
				CCP3_4	I/O	PCA 的捕获输入和脉冲输出
				INT4	I	外部中断 4
28	20	19	22	P3.1	I/O	标准 IO 口
				TxD	O	串口 1 的发送脚
				CCP2_4	I/O	PCA 的捕获输入和脉冲输出
29	21	20	23	P3.2	I/O	标准 IO 口
				INT0	I	外部中断 0
				CCP1_4	I/O	PCA 的捕获输入和脉冲输出
				SCLK_4	I/O	SPI 的时钟脚
				SCL_4	I/O	I2C 的时钟线
30	22	21	24	P3.3	I/O	标准 IO 口
				INT1	I	外部中断 1
				CCP0_4	I/O	PCA 的捕获输入和脉冲输出
				MISO_4	I/O	SPI 主机输入从机输出
				SDA_4	I/O	I2C 接口的数据线
31	23	22	25	P3.4	I/O	标准 IO 口
				T0	I	定时器 0 外部时钟输入
				T1CLKO	O	定时器 1 时钟分频输出
				MOSI_4	I/O	SPI 主机输出从机输入
				CMPO	O	比较器输出

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
32	24	23	26	P3.5	I/O	标准 IO 口
				T1	I	定时器 1 外部时钟输入
				T0CLKO	O	定时器 0 时钟分频输出
				ECI_4	I	PCA 的外部脉冲输入
				SS_4	I	SPI 的从机选择脚（主机为输出）
				PWMFLT	I	增强 PWM 的外部异常检测脚
33	25			P5.0	I/O	标准 IO 口
34	26			P5.1	I/O	标准 IO 口
35	27	24	27	P3.6	I/O	标准 IO 口
				INT2	I	外部中断 2
				WR_2	O	外部总线的写信号线
				RxD_2	I	串口 1 的接收脚
				CMP-	I	比较器负极输入
36	28	25	28	P3.7	I/O	标准 IO 口
				INT3	I	外部中断 3
				RD_2	O	外部总线的读信号线
				TxD_2	O	串口 1 的发送脚
				CMP+	I	比较器正极输入
37	29	26	29	P4.1	I/O	标准 IO 口
				ALE	O	地址锁存信号
				CMPO_2	O	比较器输出
38				P7.0	I/O	标准 IO 口
				CCP0_3	I/O	PCA 的捕获输入和脉冲输出
39				P7.1	I/O	标准 IO 口
				CCP1_3	I/O	PCA 的捕获输入和脉冲输出
40				P7.2	I/O	标准 IO 口
				CCP2_3	I/O	PCA 的捕获输入和脉冲输出
41				P7.3	I/O	标准 IO 口
				CCP3_3	I/O	PCA 的捕获输入和脉冲输出
42	30	27	30	P2.0	I/O	标准 IO 口
				A8	I	地址总线
				PWM0	O	增强 PWM 通道 0 输出脚
				RSTSV	-	端口的初始电平可 ISP 下载时配置
43	31	28		P4.2	I/O	标准 IO 口
				RD_3	O	外部总线的读信号线
				TxD2_2	O	串口 2 的发送脚

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
44	32	29	31	P2.1	I/O	标准 IO 口
				A9	I	地址总线
				PWM1	O	增强 PWM 通道 1 输出脚
45	33	30	32	P2.2	I/O	标准 IO 口
				A10	I	地址总线
				PWM2	O	增强 PWM 通道 2 输出脚
				SS_2	I	SPI 的从机选择脚（主机为输出）
				ECI_2	I	PCA 的外部脉冲输入
46	34	31	33	P2.3	I/O	标准 IO 口
				A11	I	地址总线
				PWM3	O	增强 PWM 通道 3 输出脚
				MOSI_2	I/O	SPI 主机输出从机输入
				CCP0_2	I/O	PCA 的捕获输入和脉冲输出
47	35	32	34	P2.4	I/O	标准 IO 口
				A12	I	地址总线
				PWM4	O	增强 PWM 通道 4 输出脚
				MISO_2	I/O	SPI 主机输入从机输出
				SDA_2	I/O	I2C 接口的数据线
				CCP1_2	I/O	PCA 的捕获输入和脉冲输出
48	36	33	35	P2.5	I/O	标准 IO 口
				A13	I	地址总线
				PWM5	O	增强 PWM 通道 5 输出脚
				SCLK_2	I/O	SPI 的时钟脚
				SCL_2	I/O	I2C 的时钟线
				CCP2_2	I/O	PCA 的捕获输入和脉冲输出
49	37	34	36	P2.6	I/O	标准 IO 口
				A14	I	地址总线
				PWM6	O	增强 PWM 通道 6 输出脚
				CCP3_2	I/O	PCA 的捕获输入和脉冲输出
50	38	35	37	P2.7	I/O	标准 IO 口
				A15	I	地址总线
				PWM7	O	增强 PWM 通道 7 输出脚
51	39	36	38	P0.0	I/O	标准 IO 口
				AD0	I	地址总线
				ADC8	I	ADC 模拟输入通道 8

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
52	40	37	39	P0.1	I/O	标准 IO 口
				AD1	I	地址总线
				ADC9	I	ADC 模拟输入通道 9
53	41	38	40	P0.2	I/O	标准 IO 口
				AD2	I	地址总线
				ADC10	I	ADC 模拟输入通道 10
54				P7.4	I/O	标准 IO 口
				SS_3	I	SPI 的从机选择脚（主机为输出）
				ECI_3	I	PCA 的外部脉冲输入
55				P7.5	I/O	标准 IO 口
				MOSI_3	I/O	SPI 主机输出从机输入
56				P7.6	I/O	标准 IO 口
				MISO_3	I/O	SPI 主机输入从机输出
				SDA_3	I/O	I2C 接口的数据线
57				P7.7	I/O	标准 IO 口
				SCLK_3	I/O	SPI 的时钟脚
				SCL_3	I/O	I2C 的时钟线
58	42	39		P4.3	I/O	标准 IO 口
				WR	O	外部总线的写信号线
				RxD_4	I	串口 1 的接收脚
59	43	40	1	P0.3	I/O	标准 IO 口
				AD3	I	地址总线
				ADC11	I	ADC 模拟输入通道 11
60	44	41	2	P0.4	I/O	标准 IO 口
				AD4	I	地址总线
				ADC12	I	ADC 模拟输入通道 12
				T3	I	定时器 3 外部时钟输入

编号				名称	类型	说明
LQFP64S	LQFP48	LQFP44	PDIP40			
61	45	42	3	P0.5	I/O	标准 IO 口
				AD5	I	地址总线
				ADC13	I	ADC 模拟输入通道 13
				T3CLKO	O	定时器 3 时钟分频输出
62	46	43	4	P0.6	I/O	标准 IO 口
				AD6	I	地址总线
				ADC14	I	ADC 模拟输入通道 14
				T4	I	定时器 4 外部时钟输入
63	47	44	5	P0.7	I/O	标准 IO 口
				AD7	I	地址总线
				T4CLKO	O	定时器 4 时钟分频输出
64	48	1	6	P1.0	I/O	标准 IO 口
				ADC0	I	ADC 模拟输入通道 0
				PWM0_2	O	增强 PWM 通道 0 输出脚
				RxD2	I	串口 2 的接收脚

STC MCU

2.3 STC8F2K64S4-44Pin系列

2.3.1 特性及价格

- 选型价格（不需要外部晶振、不需要外部复位）

单片机型号	工作电压（V）	Flash 程序存储器 10 万次 字节	大容量扩展 SRAM 字节	强大的双 DPTR 可增可减	EEPROM 10 万次 字节	I/O 口最多数量	串口并可掉电唤醒	SPI	I²C	定时器/计数器（T0-T4 外部管脚也可掉电唤醒）	16 位高级 PWM 定时器 互补对称死区	15 位增强型 PWM（带死区控制）	PCA/CCP/PWM（可当外部中断并可掉电唤醒）	掉电唤醒专用定时器	15 路高速 ADC(8 路 PWM 可当 8 路 D/A 使用)	比较器（可当 1 路 A/D，可作外部掉电检测）	内部低压检测中断并可掉电唤醒	看门狗 复位定时器	内部高可靠复位（可选复位门槛电压）	内部高精准时钟（24MHz 可调）	可对外输出时钟及复位	程序加密后传输（防拦截）	可设置下次更新程序需口令	支持 RS485 下载	支持 USB 直接下载	本身就可在线仿真	封装						2017 年新品供货信息
																											LQFP44	LQFP32	QFN32 <4x4mm>	TSSOP20	SOP16	SOP8	PDIP40	
STC8F2K16S4	2.0-5.5	16K	2K	2	48K	42	4	有	有	5	-	-	-	有	-	有	有	有	4 级	有	是	有	是	是	是	是	¥2.7	¥2.6						开始供货
STC8F2K32S4	2.0-5.5	32K	2K	2	32K	42	4	有	有	5	-	-	-	有	-	有	有	有	4 级	有	是	有	是	是	是	是	¥2.8	¥2.7						
STC8F2K60S4	2.0-5.5	60K	2K	2	4K	42	4	有	有	5	-	-	-	有	-	有	有	有	4 级	有	是	有	是	是	是	是	¥2.9	¥2.8						
STC8F2K64S4	2.0-5.5	64K	2K	2	IAP	42	4	有	有	5	-	-	-	有	-	有	有	有	4 级	有	是	有	是	是	是	是	¥2.9	¥2.8						

注：以上的单价为 10K 及以上订货量的价格，量小则每片需增加 0.1 元人民币。当订货的总额达到或高于 3000 元时，可免运费发货，否则需要由客户承担运费。零售 10 片起售。

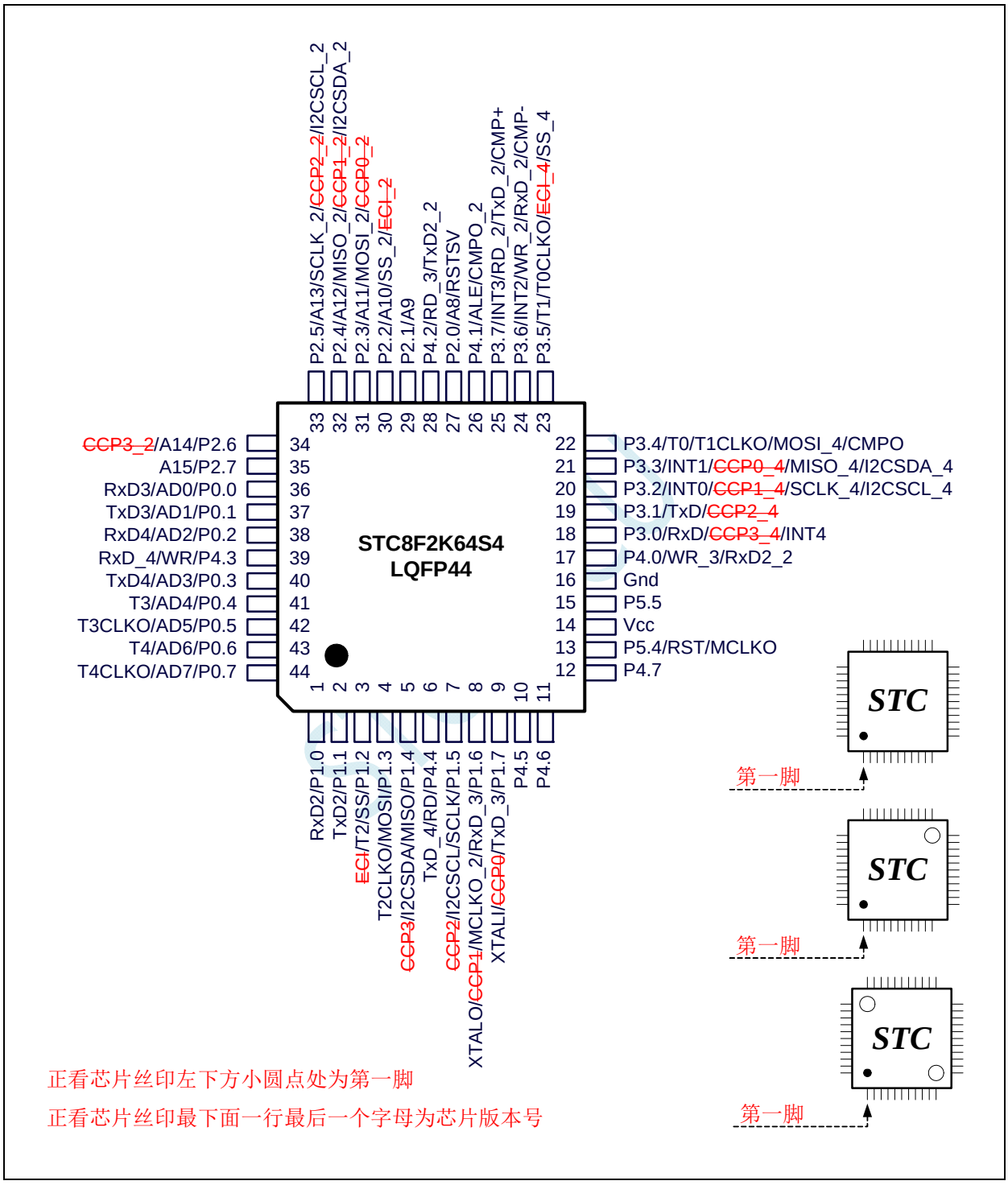
- **内核**
  - ✓ 超高速 8051 内核（1T），比传统 8051 约快 12 倍以上
  - ✓ 指令代码完全兼容传统 8051
  - ✓ 18 个中断源，4 级中断优先级
  - ✓ 支持在线仿真
- **工作电压**
  - ✓ 2.0V～5.5V
  - ✓ 内建 LDO
- **工作温度**
  - ✓ -40℃～85℃（超温度范围应用请参考电器特性章节说明）
- **Flash 存储器**
  - ✓ 最大 64K 字节 FLASH 程序存储器（ROM），用于存储用户代码
  - ✓ 支持用户配置 EEPROM 大小，512 字节单页擦除，擦写次数可达 10 万次以上
  - ✓ 支持在系统编程方式（ISP）更新用户应用程序，无需专用编程器
  - ✓ 支持单芯片仿真，无需专用仿真器，理论断点个数无限制
- **SRAM**
  - ✓ 128 字节内部直接访问 RAM（DATA）
  - ✓ 128 字节内部间接访问 RAM（IDATA）
  - ✓ 2048 字节内部扩展 RAM（内部 XDATA）
  - ✓ 外部最大可扩展 64K 字节 RAM（外部 XDATA）

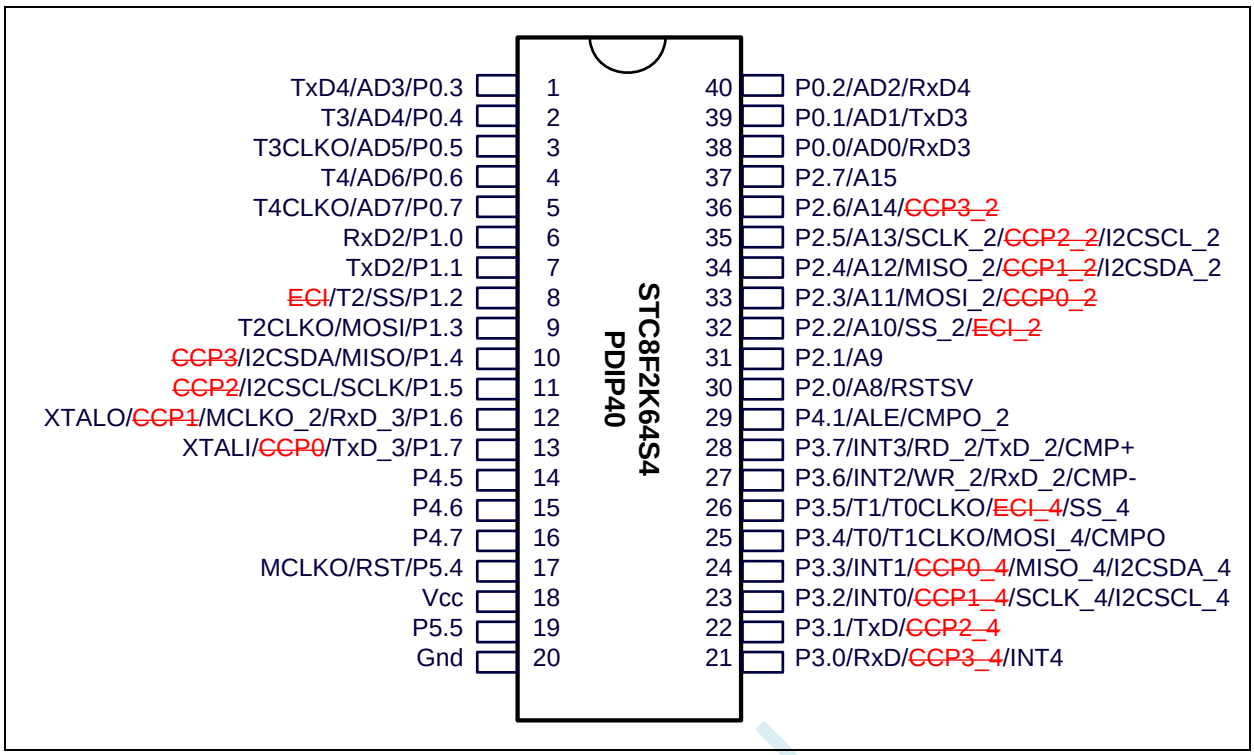
- **时钟控制**
  - ✓ 内部 24MHz 高精度 IRC（ISP 编程时可进行上下调整）
    - ⌖ 误差±0.3%（常温下 25℃）
    - ⌖ -1.8%～+0.8%温漂（全温度范围，-40℃～85℃）
    - ⌖ -1.0%～+0.5%温漂（温度范围，-20℃～65℃）
  - ✓ 内部 32KHz 低速 IRC（误差较大）
  - ✓ 外部晶振（4MHz～33MHz）和外部时钟

    用户可自由选择上面的 3 种时钟源
- **复位**
  - ✓ 硬件复位
    - ⌖ 上电复位
    - ⌖ 复位脚复位（高电平复位），出厂时 P5.4 默认为 IO 口，ISP 下载时可将 P5.4 管脚设置为复位脚
    - ⌖ 看门狗溢出复位
    - ⌖ 低压检测复位，提供 4 级低压检测电压：2.2V、2.4V、2.7V、3.0V
  - ✓ 软件复位
    - ⌖ 软件方式写复位触发寄存器
- **中断**
  - ✓ 提供 18 个中断源：INT0、INT1、INT2、INT3、INT4、定时器 0、定时器 1、定时器 2、定时器 3、定时器 4、串口 1、串口 2、串口 3、串口 4、LVD 低压检测、SPI、I^2C、比较器
  - ✓ 提供 4 级中断优先级
- **数字外设**
  - ✓ 5 个 16 位定时器：定时器 0、定时器 1、定时器 2、定时器 3、定时器 4，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式
  - ✓ 4 个高速串口：串口 1、串口 2、串口 3、串口 4，波特率时钟源最快可为 FOSC/4
  - ✓ SPI：支持主机模式和从机模式以及主机/从机自动切换
  - ✓ I^2C：支持主机模式和从机模式
- **模拟外设**
  - ✓ 比较器，一组比较器
- **GPIO**
  - ✓ 最多可达 42 个 GPIO：P0.0~P0.7、P1.0~P1.7、P2.0~P2.7、P3.0~P3.7、P4.0~P4.7、P5.4~P5.5
  - ✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式
- **封装**
  - ✓ LQFP44、LQFP32、~~PDIP40（暂未生产）~~

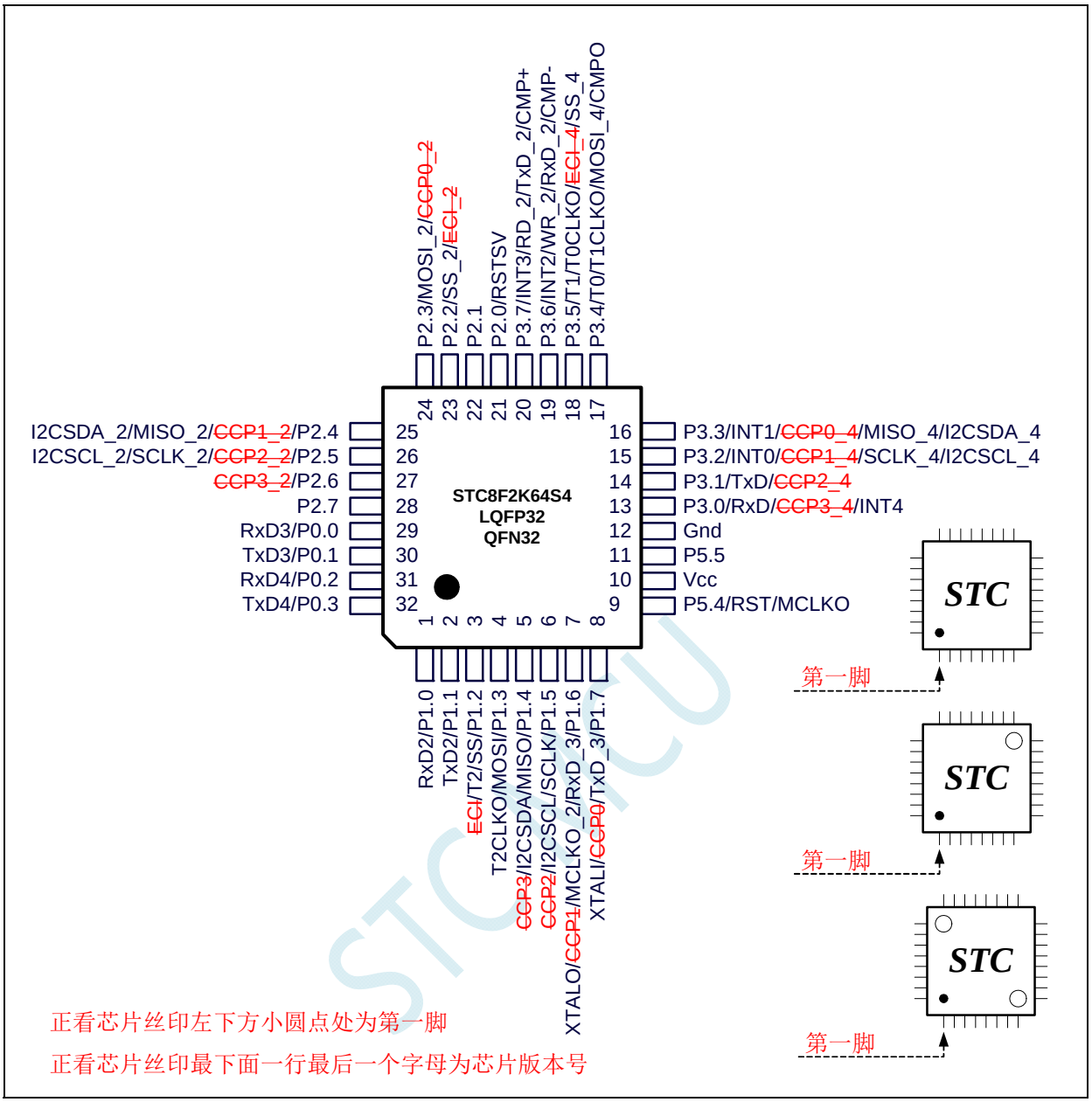
STCMCU

2.3.2 管脚图


P2.5/A13/SCLK_2/~~CCP2_2~~/I2CSCL_2
P2.4/A12/MISO_2/~~CCP1_2~~/I2CSDA_2
P2.3/A11/MOSI_2/~~CCP0_2~~
P2.2/A10/SS_2/~~ECI_2~~
P2.1/A9
P4.2/RD_3/TxD2_2
P2.0/A8/RSTSV
P4.1/ALE/CMPO_2
P3.7/INT3/RD_2/TxD_2/CMP+
P3.6/INT2/WR_2/RxD_2/CMP-
P3.5/T1/T0CLKO/~~ECI_4~~/SS_4
~~CCP3_2~~/A14/P2.6
A15/P2.7
RxD3/AD0/P0.0
TxD3/AD1/P0.1
RxD4/AD2/P0.2
RxD_4/WR/P4.3
TxD4/AD3/P0.3
T3/AD4/P0.4
T3CLKO/AD5/P0.5
T4/AD6/P0.6
T4CLKO/AD7/P0.7
STC8F2K64S4
LQFP44
P3.4/T0/T1CLKO/MOSI_4/CMPO
P3.3/INT1/~~CCP0_4~~/MISO_4/I2CSDA_4
P3.2/INT0/~~CCP1_4~~/SCLK_4/I2CSCL_4
P3.1/TxD/~~CCP2_4~~
P3.0/RxD/~~CCP3_4~~/INT4
P4.0/WR_3/RxD2_2
Gnd
P5.5
Vcc
P5.4/RST/MCLKO
P4.7
RxD2/P1.0
TxD2/P1.1
~~ECI~~/T2/SS/P1.2
T2CLKO/MOSI/P1.3
~~CCP3~~/I2CSDA/MISO/P1.4
TxD_4/RD/P4.4
~~CCP2~~/I2CSCL/SCLK/P1.5
~~CCP1~~/MCLKO_2/RxD_3/P1.6
XTALI/~~CCP0~~/TxD_3/P1.7
P4.5
P4.6
XTALO
STC
第一脚
STC
第一脚
STC
第一脚
正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号


STC8F2K64S4
PDIP40
1 TxD4/AD3/P0.3
2 T3/AD4/P0.4
3 T3CLKO/AD5/P0.5
4 T4/AD6/P0.6
5 T4CLKO/AD7/P0.7
6 RxD2/P1.0
7 TxD2/P1.1
8 ~~ECI~~/T2/SS/P1.2
9 T2CLKO/MOSI/P1.3
10 ~~CCP3~~/I2CSDA/MISO/P1.4
11 ~~CCP2~~/I2CSCL/SCLK/P1.5
12 XTALO/~~CCP1~~/MCLKO_2/RxD_3/P1.6
13 XTALI/~~CCP0~~/TxD_3/P1.7
14 P4.5
15 P4.6
16 P4.7
17 MCLKO/RST/P5.4
18 Vcc
19 P5.5
20 Gnd
40 P0.2/AD2/RxD4
39 P0.1/AD1/TxD3
38 P0.0/AD0/RxD3
37 P2.7/A15
36 P2.6/A14/~~CCP3_2~~
35 P2.5/A13/SCLK_2/~~CCP2_2~~/I2CSCL_2
34 P2.4/A12/MISO_2/~~CCP1_2~~/I2CSDA_2
33 P2.3/A11/MOSI_2/~~CCP0_2~~
32 P2.2/A10/SS_2/~~ECI_2~~
31 P2.1/A9
30 P2.0/A8/RSTSV
29 P4.1/ALE/CMPO_2
28 P3.7/INT3/RD_2/TxD_2/CMP+
27 P3.6/INT2/WR_2/RxD_2/CMP-
26 P3.5/T1/T0CLKO/~~ECI_4~~/SS_4
25 P3.4/T0/T1CLKO/MOSI_4/CMPO
24 P3.3/INT1/~~CCP0_4~~/MISO_4/I2CSDA_4
23 P3.2/INT0/~~CCP1_4~~/SCLK_4/I2CSCL_4
22 P3.1/TxD/~~CCP2_4~~
21 P3.0/RxD/~~CCP3_4~~/INT4

STC MCU


P2.3/MOSI_2/~~CCP0_2~~
P2.2/SS_2/~~ECI_2~~
P2.1
P2.0/RSTSV
P3.7/INT3/RD_2/TxD_2/CMP+
P3.6/INT2/WR_2/RxD_2/CMP-
P3.5/T1/T0CLKO/~~ECI_4~~/SS_4
P3.4/T0/T1CLKO/MOSI_4/CMPO
24 23 22 21 20 19 18 17
I2CSDA_2/MISO_2/~~CCP1_2~~/P2.4 25
I2CSCL_2/SCLK_2/~~CCP2_2~~/P2.5 26
~~CCP3_2~~/P2.6 27
P2.7 28
RxD3/P0.0 29
TxD3/P0.1 30
RxD4/P0.2 31
TxD4/P0.3 32
STC8F2K64S4
LQFP32
QFN32
16 P3.3/INT1/~~CCP0_4~~/MISO_4/I2CSDA_4
15 P3.2/INT0/~~CCP1_4~~/SCLK_4/I2CSCL_4
14 P3.1/TxD/~~CCP2_4~~
13 P3.0/RxD/~~CCP3_4~~/INT4
12 Gnd
11 P5.5
10 Vcc
9 P5.4/RST/MCLKO
1 2 3 4 5 6 7 8
RxD2/P1.0
TxD2/P1.1
~~ECI~~/T2/SS/P1.2
T2CLKO/MOSI/P1.3
~~CCP3~~/I2CSDA/MISO/P1.4
~~CCP2~~/I2CSCL/SCLK/P1.5
XTALO/~~CCP1~~/MCLKO_2/RxD_3/P1.6
XTALI/~~CCP0~~/TxD_3/P1.7
STC
第一脚
STC
第一脚
STC
第一脚
正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号

2.3.3 管脚说明

编号			名称	类型	说明
LQFP44	PDIP40	LQFP32			
2	7	2	P1.1	I/O	标准 IO 口
			TxD2	O	串口 2 的发送脚
3	8	3	P1.2	I/O	标准 IO 口
			SS	I	SPI 的从机选择脚（主机为输出）
			T2	I	定时器 2 外部时钟输入
4	9	4	P1.3	I/O	标准 IO 口
			MOSI	I/O	SPI 主机输出从机输入
			T2CLKO	O	定时器 2 时钟分频输出
5	10	5	P1.4	I/O	标准 IO 口
			MISO	I/O	SPI 主机输入从机输出
			SDA	I/O	I2C 接口的数据线
6			P4.4	I/O	标准 IO 口
			RD	O	外部总线的读信号线
			TxD_4	O	串口 1 的发送脚
7	11	6	P1.5	I/O	标准 IO 口
			SCLK	I/O	SPI 的时钟脚
			SCL	I/O	I2C 的时钟线
8	12	7	P1.6	I/O	标准 IO 口
			RxD_3	I	串口 1 的接收脚
			XTALO	O	外部晶振的输出脚
			MCLKO_2	O	主时钟分频输出
9	13	8	P1.7	I/O	标准 IO 口
			TxD_3	O	串口 1 的发送脚
			XTALI	I	外部晶振/外部时钟的输入脚
10	14		P4.5	I/O	标准 IO 口
11	15		P4.6	I/O	标准 IO 口
12	16		P4.7	I/O	标准 IO 口
13	17	9	P5.4	I/O	标准 IO 口
			RST	I	复位引脚
			MCLKO	O	主时钟分频输出
14	18	10	Vcc	VCC	电源脚

编号			名称	类型	说明
LQFP44	PDIP40	LQFP32			
15	19	11	P5.5	I/O	标准 IO 口
16	20	12	Gnd	GND	地线
17			P4.0	I/O	标准 IO 口
			WR_3	O	外部总线的写信号线
			RxD2_2	I	串口 2 的接收脚
18	21	13	P3.0	I/O	标准 IO 口
			RxD	I	串口 1 的接收脚
			INT4	I	外部中断 4
19	22	14	P3.1	I/O	标准 IO 口
			TxD	O	串口 1 的发送脚
20	23	15	P3.2	I/O	标准 IO 口
			INT0	I	外部中断 0
			SCL_4	I/O	I2C 的时钟线
			SCLK_4	I/O	SPI 的时钟脚
21	24	16	P3.3	I/O	标准 IO 口
			INT1	I	外部中断 1
			SDA_4	I/O	I2C 接口的数据线
			MISO_4	I/O	SPI 主机输入从机输出
22	25	17	P3.4	I/O	标准 IO 口
			T0	I	定时器 0 外部时钟输入
			T1CLKO	O	定时器 1 时钟分频输出
			MOSI_4	I/O	SPI 主机输出从机输入
			CMPO	O	比较器输出
23	26	18	P3.5	I/O	标准 IO 口
			T1	I	定时器 1 外部时钟输入
			T0CLKO	O	定时器 0 时钟分频输出
			SS_4	I	SPI 的从机选择脚（主机为输出）
24	27	19	P3.6	I/O	标准 IO 口
			INT2	I	外部中断 2
			WR_2	O	外部总线的写信号线
			RxD_2	I	串口 1 的接收脚
			CMP-	I	比较器负极输入

编号			名称	类型	说明
LQFP44	PDIP40	LQFP32			
25	28	20	P3.7	I/O	标准 IO 口
			INT3	I	外部中断 3
			RD_2	O	外部总线的读信号线
			TxD_2	O	串口 1 的发送脚
			CMP+	I	比较器正极输入
26	29		P4.1	I/O	标准 IO 口
			ALE	O	地址锁存信号
			CMPO_2	O	比较器输出
27	30	21	P2.0	I/O	标准 IO 口
			A8	I	地址总线
			RSTSV	-	端口的初始电平可 ISP 下载时配置
28			P4.2	I/O	标准 IO 口
			RD_3	O	外部总线的读信号线
			TxD2_2	O	串口 2 的发送脚
29	31	22	P2.1	I/O	标准 IO 口
			A9	I	地址总线
30	32	23	P2.2	I/O	标准 IO 口
			A10	I	地址总线
			SS_2	I	SPI 的从机选择脚（主机为输出）
31	33	24	P2.3	I/O	标准 IO 口
			A11	I	地址总线
			MOSI_2	I/O	SPI 主机输出从机输入
32	34	25	P2.4	I/O	标准 IO 口
			A12	I	地址总线
			MISO_2	I/O	SPI 主机输入从机输出
			SDA_2	I/O	I2C 接口的数据线
33	35	26	P2.5	I/O	标准 IO 口
			A13	I	地址总线
			SCLK_2	I/O	SPI 的时钟脚
			SCL_2	I/O	I2C 的时钟线
34	36	27	P2.6	I/O	标准 IO 口
			A14	I	地址总线
35	37	28	P2.7	I/O	标准 IO 口
			A15	I	地址总线

编号			名称	类型	说明
LQFP44	PDIP40	LQFP32			
36	38	29	P0.0	I/O	标准 IO 口
			AD0	I	地址总线
			RxD3	I	串口 3 的接收脚
37	39	30	P0.1	I/O	标准 IO 口
			AD1	I	地址总线
			TxD3	O	串口 3 的发送脚
38	40	31	P0.2	I/O	标准 IO 口
			AD2	I	地址总线
			RxD4	I	串口 4 的接收脚
39			P4.3	I/O	标准 IO 口
			WR	O	外部总线的写信号线
			RxD_4	I	串口 1 的接收脚
40	1	32	P0.3	I/O	标准 IO 口
			AD3	I	地址总线
			TxD4	O	串口 4 的发送脚
41	2		P0.4	I/O	标准 IO 口
			AD4	I	地址总线
			T3	I	定时器 3 外部时钟输入
42	3		P0.5	I/O	标准 IO 口
			AD5	I	地址总线
			T3CLKO	O	定时器 3 时钟分频输出
43	4		P0.6	I/O	标准 IO 口
			AD6	I	地址总线
			T4	I	定时器 4 外部时钟输入
44	5		P0.7	I/O	标准 IO 口
			AD7	I	地址总线
			T4CLKO	O	定时器 4 时钟分频输出
1	6	1	P1.0	I/O	标准 IO 口
			RxD2	I	串口 2 的接收脚

2.4 STC8F2K64S2-44Pin系列

2.4.1 特性及价格

- 选型价格（不需要外部晶振、不需要外部复位）

单片机型号	工作电压（V）	Flash 程序存储器 10 万次 字节	大容量扩展 SRAM 字节	强大的双 DPTR 可增可减	EEPROM 10 万次 字节	I/O 口最多数量	串口并可掉电唤醒	SPI	I²C	定时器/计数器（T0-T4 外部管脚也可掉电唤醒）	16 位高级 PWM 定时器 互补对称死区	15 位增强型 PWM（带死区控制）	PCA/CCP/PWM（可当外部中断并可掉电唤醒）	掉电唤醒专用定时器	15 路高速 ADC(8 路 PWM 可当 8 路 D/A 使用)	比较器（可当 1 路 A/D，可作外部掉电检测）	内部低压检测中断并可掉电唤醒	看门狗 复位定时器	内部高可靠复位（可选复位门槛电压）	内部高精准时钟（24MHz 可调）	可对外输出时钟及复位	程序加密后传输（防拦截）	可设置下次更新程序需口令	支持 RS485 下载	支持 USB 直接下载	本身就可在线仿真	封装							2017 年新品供货信息
																											LQFP44	LQFP32	QFN32 <4x4mm>	TSSOP20	SOP16	SOP8	PDIP40	
~~STC8F2K08S2~~	~~2.0-5.5~~	~~8K~~	~~2K~~	~~2~~	~~4K~~	~~18~~	~~2~~	~~有~~	~~有~~	~~5~~	~~-~~	~~-~~	~~-~~	~~有~~	~~-~~	~~有~~	~~有~~	~~有~~	~~4 级~~	~~有~~	~~是~~	~~有~~	~~是~~	~~是~~	~~是~~	~~是~~				~~¥1.4~~	~~1.35~~			大量供货
STC8F2K16S2	2.0-5.5	16K	2K	2	4K	42	2	有	有	5	-	-	-	有	-	有	有	有	4 级	有	是	有	是	是	是	是	¥1.8	¥1.6	¥1.6	¥1.4	1.35			
STC8F2K32S2	2.0-5.5	32K	2K	2	32K	42	2	有	有	5	-	-	-	有	-	有	有	有	4 级	有	是	有	是	是	是	是	1.99	¥1.8	¥1.8					
STC8F2K60S2	2.0-5.5	60K	2K	2	4K	42	2	有	有	5	-	-	-	有	-	有	有	有	4 级	有	是	有	是	是	是	是	¥2.2	¥2.0						
STC8F2K64S2	2.0-5.5	64K	2K	2	IAP	42	2	有	有	5	-	-	-	有	-	有	有	有	4 级	有	是	有	是	是	是	是	¥2.2	¥2.0						

注：以上的单价为 10K 及以上订货量的价格，量小则每片需增加 0.1 元人民币。当订货的总额达到或高于 3000 元时，可免运费发货，否则需要由客户承担运费。零售 10 片起售。

- **内核**
  - ✓ 超高速 8051 内核（1T），比传统 8051 约快 12 倍以上
  - ✓ 指令代码完全兼容传统 8051
  - ✓ 16 个中断源，4 级中断优先级
  - ✓ 支持在线仿真
- **工作电压**
  - ✓ 2.0V～5.5V
  - ✓ 内建 LDO
- **工作温度**
  - ✓ -40℃～85℃（超温度范围应用请参考电器特性章节说明）
- **Flash 存储器**
  - ✓ 最大 64K 字节 FLASH 程序存储器（ROM），用于存储用户代码
  - ✓ 支持用户配置 EEPROM 大小，512 字节单页擦除，擦写次数可达 10 万次以上
  - ✓ 支持在系统编程方式（ISP）更新用户应用程序，无需专用编程器
  - ✓ 支持单芯片仿真，无需专用仿真器，理论断点个数无限制
- **SRAM**
  - ✓ 128 字节内部直接访问 RAM（DATA）
  - ✓ 128 字节内部间接访问 RAM（IDATA）
  - ✓ 2048 字节内部扩展 RAM（内部 XDATA）

- ✓ 外部最大可扩展 64K 字节 RAM（外部 XDATA）

- **时钟控制**
  - ✓ 内部 24MHz 高精度 IRC（ISP 编程时可进行上下调整）
    - ⌖ 误差±0.3%（常温下 25℃）
    - ⌖ -1.8%～+0.8%温漂（全温度范围，-40℃～85℃）
    - ⌖ -1.0%～+0.5%温漂（温度范围，-20℃～65℃）
  - ✓ 内部 32KHz 低速 IRC（误差较大）
  - ✓ 外部晶振（4MHz～33MHz）和外部时钟
    用户可自由选择上面的 3 种时钟源

- **复位**
  - ✓ 硬件复位
    - ⌖ 上电复位
    - ⌖ 复位脚复位（高电平复位），出厂时 P5.4 默认为 IO 口，ISP 下载时可将 P5.4 管脚设置为复位脚
    - ⌖ 看门狗溢出复位
    - ⌖ 低压检测复位，提供 4 级低压检测电压：2.2V、2.4V、2.7V、3.0V
  - ✓ 软件复位
    - ⌖ 软件方式写复位触发寄存器

- **中断**
  - ✓ 提供 16 个中断源：INT0、INT1、INT2、INT3、INT4、定时器 0、定时器 1、定时器 2、定时器 3、定时器 4、串口 1、串口 2、LVD 低压检测、SPI、I^2C、比较器
  - ✓ 提供 4 级中断优先级

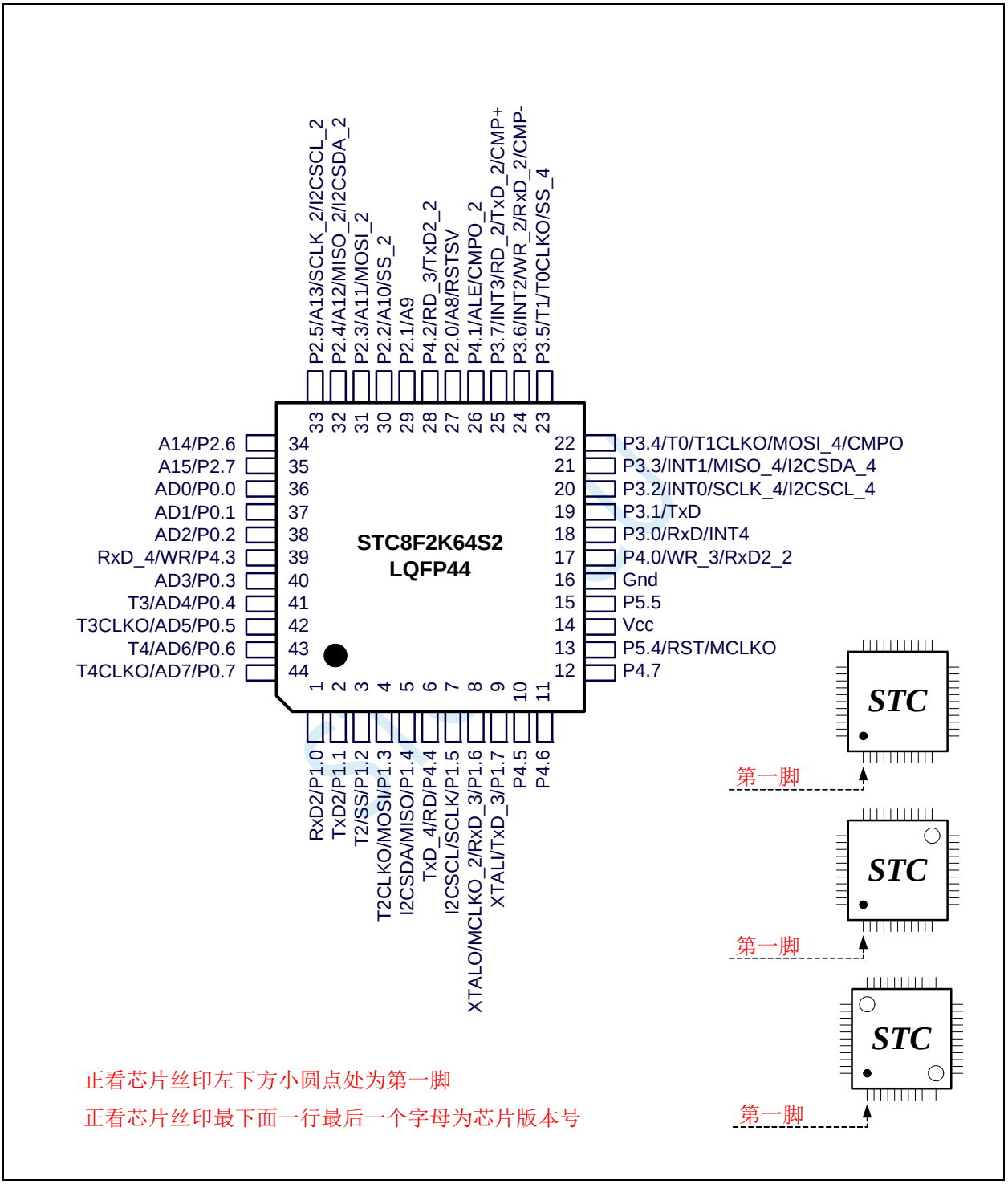
- **数字外设**
  - ✓ 5 个 16 位定时器：定时器 0、定时器 1、定时器 2、定时器 3、定时器 4，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式
  - ✓ 2 个高速串口：串口 1、串口 2，波特率时钟源最快可为 FOSC/4
  - ✓ SPI：支持主机模式和从机模式以及主机/从机自动切换
  - ✓ I^2C：支持主机模式和从机模式

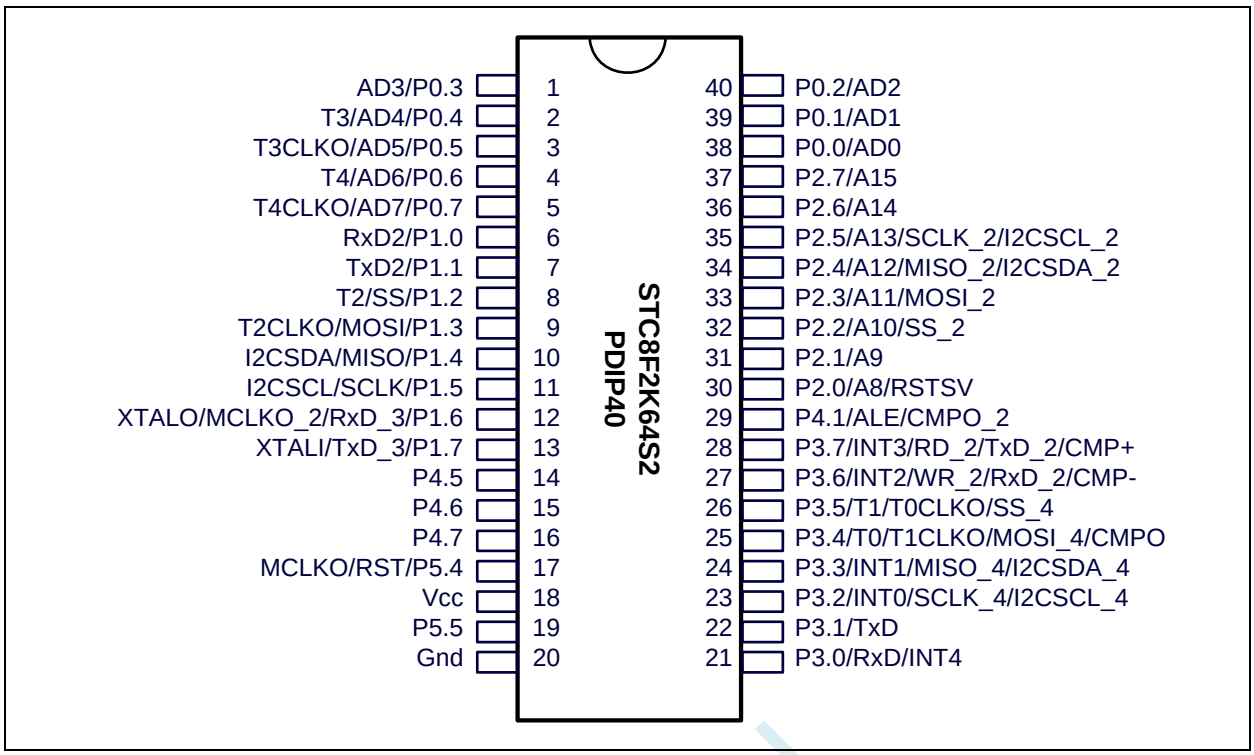
- **模拟外设**
  - ✓ 比较器，一组比较器

- **GPIO**
  - ✓ 最多可达 42 个 GPIO：P0.0~P0.7、P1.0~P1.7、P2.0~P2.7、P3.0~P3.7、P4.0~P4.7、P5.4~P5.5
  - ✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式

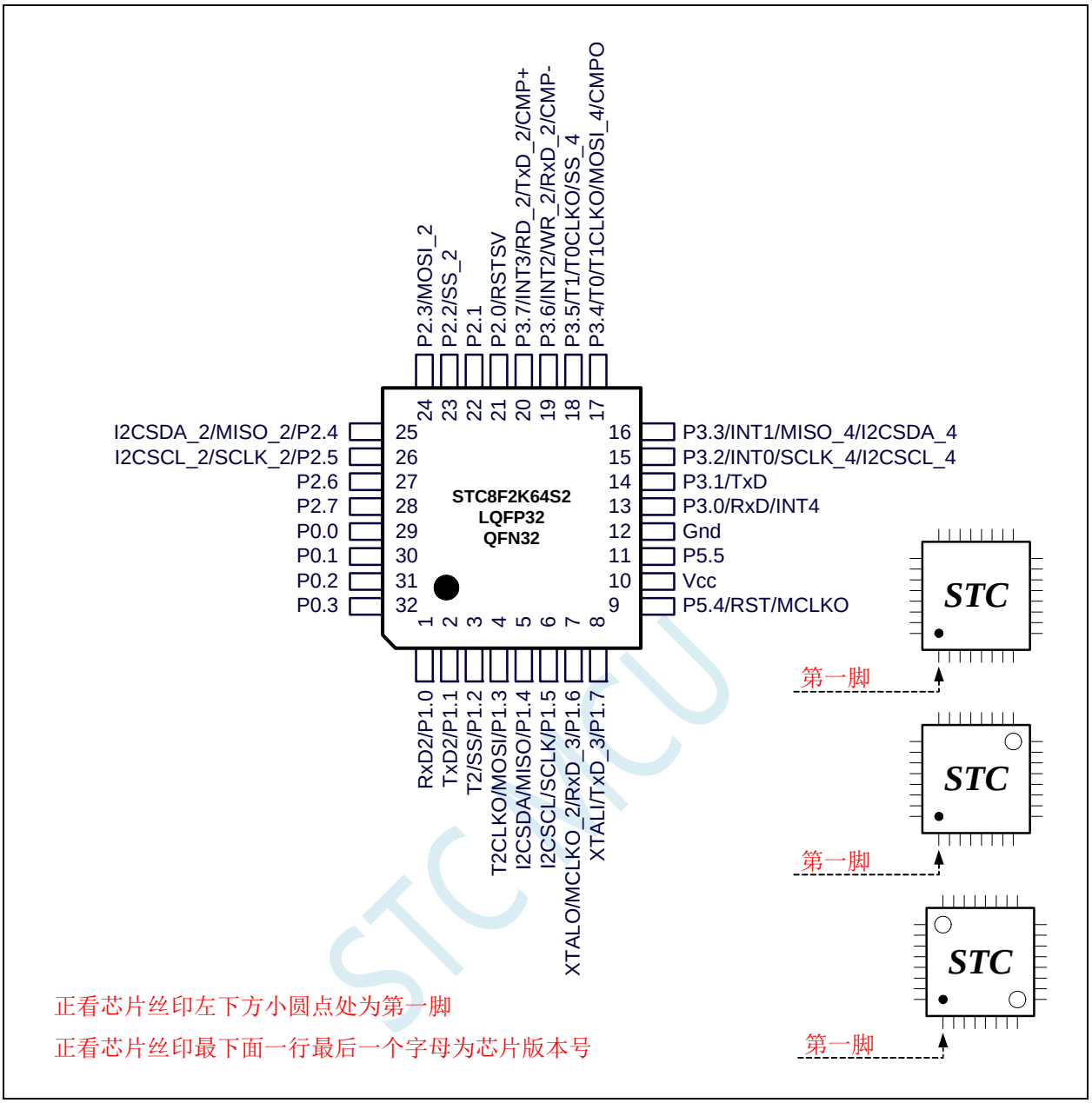
- **封装**
  - ✓ LQFP44、LQFP32、PDIP40、TSSOP20、SOP16

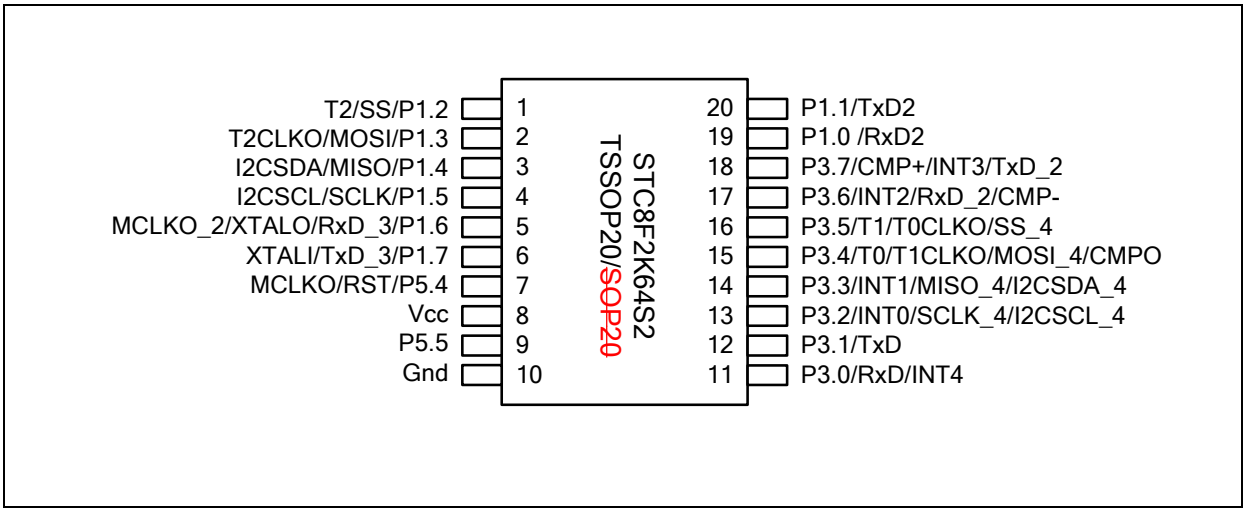
2.4.2 管脚图

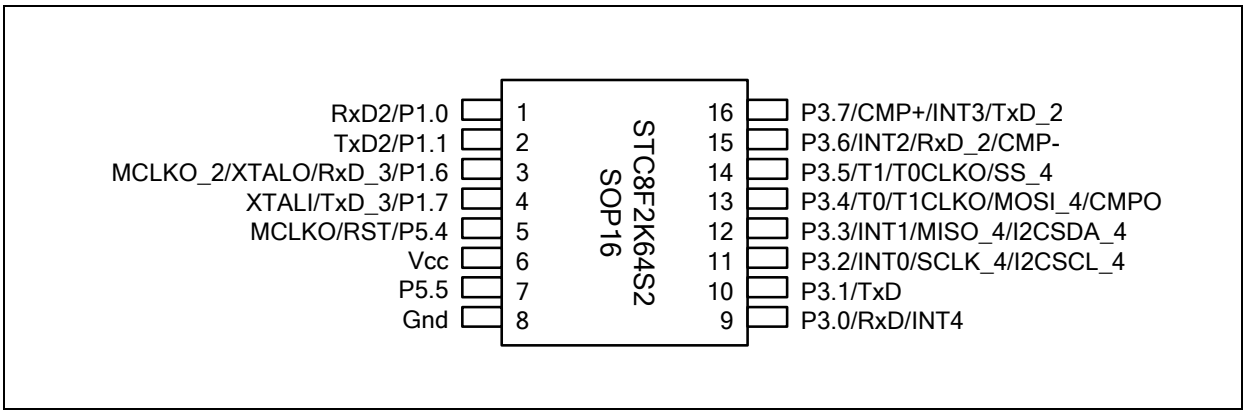

P2.5/A13/SCLK_2/I2CSCL_2
P2.4/A12/MISO_2/I2CSDA_2
P2.3/A11/MOSI_2
P2.2/A10/SS_2
P2.1/A9
P4.2/RD_3/TxD2_2
P2.0/A8/RSTSV
P4.1/ALE/CMPO_2
P3.7/INT3/RD_2/TxD_2/CMP+
P3.6/INT2/WR_2/RxD_2/CMP-
P3.5/T1/T0CLKO/SS_4
33 32 31 30 29 28 27 26 25 24 23
A14/P2.6 34
A15/P2.7 35
AD0/P0.0 36
AD1/P0.1 37
AD2/P0.2 38
RxD_4/WR/P4.3 39
AD3/P0.3 40
T3/AD4/P0.4 41
T3CLKO/AD5/P0.5 42
T4/AD6/P0.6 43
T4CLKO/AD7/P0.7 44
STC8F2K64S2
LQFP44
22 P3.4/T0/T1CLKO/MOSI_4/CMPO
21 P3.3/INT1/MISO_4/I2CSDA_4
20 P3.2/INT0/SCLK_4/I2CSCL_4
19 P3.1/TxD
18 P3.0/RxD/INT4
17 P4.0/WR_3/RxD2_2
16 Gnd
15 P5.5
14 Vcc
13 P5.4/RST/MCLKO
12 P4.7
1 2 3 4 5 6 7 8 9 10 11
RxD2/P1.0
TxD2/P1.1
T2/SS/P1.2
T2CLKO/MOSI/P1.3
I2CSDA/MISO/P1.4
TxD_4/RD/P4.4
I2CSCL/SCLK/P1.5
XTALI/TxD_3/P1.7
XTALO/MCLKO_2/RxD_3/P1.6
P4.5
P4.6
STC
第一脚
STC
第一脚
STC
第一脚
正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号


STC8F2K64S2
PDIP40
AD3/P0.3 1
T3/AD4/P0.4 2
T3CLKO/AD5/P0.5 3
T4/AD6/P0.6 4
T4CLKO/AD7/P0.7 5
RxD2/P1.0 6
TxD2/P1.1 7
T2/SS/P1.2 8
T2CLKO/MOSI/P1.3 9
I2CSDA/MISO/P1.4 10
I2CSCL/SCLK/P1.5 11
XTALO/MCLKO_2/RxD_3/P1.6 12
XTALI/TxD_3/P1.7 13
P4.5 14
P4.6 15
P4.7 16
MCLKO/RST/P5.4 17
Vcc 18
P5.5 19
Gnd 20
40 P0.2/AD2
39 P0.1/AD1
38 P0.0/AD0
37 P2.7/A15
36 P2.6/A14
35 P2.5/A13/SCLK_2/I2CSCL_2
34 P2.4/A12/MISO_2/I2CSDA_2
33 P2.3/A11/MOSI_2
32 P2.2/A10/SS_2
31 P2.1/A9
30 P2.0/A8/RSTSV
29 P4.1/ALE/CMPO_2
28 P3.7/INT3/RD_2/TxD_2/CMP+
27 P3.6/INT2/WR_2/RxD_2/CMP-
26 P3.5/T1/T0CLKO/SS_4
25 P3.4/T0/T1CLKO/MOSI_4/CMPO
24 P3.3/INT1/MISO_4/I2CSDA_4
23 P3.2/INT0/SCLK_4/I2CSCL_4
22 P3.1/TxD
21 P3.0/RxD/INT4

STC MCU


P2.3/MOSI_2
P2.2/SS_2
P2.1
P2.0/RSTSV
P3.7/INT3/RD_2/TxD_2/CMP+
P3.6/INT2/WR_2/RxD_2/CMP-
P3.5/T1/T0CLKO/SS_4
P3.4/T0/T1CLKO/MOSI_4/CMPO
24 23 22 21 20 19 18 17
I2CSDA_2/MISO_2/P2.4 25
I2CSCL_2/SCLK_2/P2.5 26
P2.6 27
P2.7 28
P0.0 29
P0.1 30
P0.2 31
P0.3 32
STC8F2K64S2
LQFP32
QFN32
16 P3.3/INT1/MISO_4/I2CSDA_4
15 P3.2/INT0/SCLK_4/I2CSCL_4
14 P3.1/TxD
13 P3.0/RxD/INT4
12 Gnd
11 P5.5
10 Vcc
9 P5.4/RST/MCLKO
1 2 3 4 5 6 7 8
RxD2/P1.0
TxD2/P1.1
T2/SS/P1.2
T2CLKO/MOSI/P1.3
I2CSDA/MISO/P1.4
I2CSCL/SCLK/P1.5
XTALO/MCLKO_2/RxD_3/P1.6
XTALI/TxD_3/P1.7
STC
第一脚
STC
第一脚
STC
第一脚
正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号


T2/SS/P1.2 1
T2CLKO/MOSI/P1.3 2
I2CSDA/MISO/P1.4 3
I2CSCL/SCLK/P1.5 4
MCLKO_2/XTALO/RxD_3/P1.6 5
XTALI/TxD_3/P1.7 6
MCLKO/RST/P5.4 7
Vcc 8
P5.5 9
Gnd 10
STC8F2K64S2
TSSOP20/SOP20
20 P1.1/TxD2
19 P1.0 /RxD2
18 P3.7/CMP+/INT3/TxD_2
17 P3.6/INT2/RxD_2/CMP-
16 P3.5/T1/T0CLKO/SS_4
15 P3.4/T0/T1CLKO/MOSI_4/CMPO
14 P3.3/INT1/MISO_4/I2CSDA_4
13 P3.2/INT0/SCLK_4/I2CSCL_4
12 P3.1/TxD
11 P3.0/RxD/INT4


RxD2/P1.0
TxD2/P1.1
MCLKO_2/XTALO/RxD_3/P1.6
XTALI/TxD_3/P1.7
MCLKO/RST/P5.4
Vcc
P5.5
Gnd
1
2
3
4
5
6
7
8
STC8F2K64S2
SOP16
16
15
14
13
12
11
10
9
P3.7/CMP+/INT3/TxD_2
P3.6/INT2/RxD_2/CMP-
P3.5/T1/T0CLKO/SS_4
P3.4/T0/T1CLKO/MOSI_4/CMPO
P3.3/INT1/MISO_4/I2CSDA_4
P3.2/INT0/SCLK_4/I2CSCL_4
P3.1/TxD
P3.0/RxD/INT4

STC MCU

2.4.3 管脚说明

编号			名称	类型	说明
LQFP44	PDIP40	LQFP32			
2	7	2	P1.1	I/O	标准 IO 口
			TxD2	O	串口 2 的发送脚
3	8	3	P1.2	I/O	标准 IO 口
			SS	I	SPI 的从机选择脚（主机为输出）
			T2	I	定时器 2 外部时钟输入
4	9	4	P1.3	I/O	标准 IO 口
			MOSI	I/O	SPI 主机输出从机输入
			T2CLKO	O	定时器 2 时钟分频输出
5	10	5	P1.4	I/O	标准 IO 口
			MISO	I/O	SPI 主机输入从机输出
			SDA	I/O	I2C 接口的数据线
6			P4.4	I/O	标准 IO 口
			RD	O	外部总线的读信号线
			TxD_4	O	串口 1 的发送脚
7	11	6	P1.5	I/O	标准 IO 口
			SCLK	I/O	SPI 的时钟脚
			SCL	I/O	I2C 的时钟线
8	12	7	P1.6	I/O	标准 IO 口
			RxD_3	I	串口 1 的接收脚
			XTALO	O	外部晶振的输出脚
			MCLKO_2	O	主时钟分频输出
9	13	8	P1.7	I/O	标准 IO 口
			TxD_3	O	串口 1 的发送脚
			XTALI	I	外部晶振/外部时钟的输入脚
10	14		P4.5	I/O	标准 IO 口
11	15		P4.6	I/O	标准 IO 口
12	16		P4.7	I/O	标准 IO 口
13	17	9	P5.4	I/O	标准 IO 口
			RST	I	复位引脚
			MCLKO	O	主时钟分频输出
14	18	10	Vcc	VCC	电源脚

编号			名称	类型	说明
LQFP44	PDIP40	LQFP32			
15	19	11	P5.5	I/O	标准 IO 口
16	20	12	Gnd	GND	地线
17			P4.0	I/O	标准 IO 口
			WR_3	O	外部总线的写信号线
			RxD2_2	I	串口 2 的接收脚
18	21	13	P3.0	I/O	标准 IO 口
			RxD	I	串口 1 的接收脚
			INT4	I	外部中断 4
19	22	14	P3.1	I/O	标准 IO 口
			TxD	O	串口 1 的发送脚
20	23	15	P3.2	I/O	标准 IO 口
			INT0	I	外部中断 0
			SCL_4	I/O	I2C 的时钟线
			SCLK_4	I/O	SPI 的时钟脚
21	24	16	P3.3	I/O	标准 IO 口
			INT1	I	外部中断 1
			SDA_4	I/O	I2C 接口的数据线
			MISO_4	I/O	SPI 主机输入从机输出
22	25	17	P3.4	I/O	标准 IO 口
			T0	I	定时器 0 外部时钟输入
			T1CLKO	O	定时器 1 时钟分频输出
			MOSI_4	I/O	SPI 主机输出从机输入
			CMPO	O	比较器输出
23	26	18	P3.5	I/O	标准 IO 口
			T1	I	定时器 1 外部时钟输入
			T0CLKO	O	定时器 0 时钟分频输出
			SS_4	I	SPI 的从机选择脚（主机为输出）
24	27	19	P3.6	I/O	标准 IO 口
			INT2	I	外部中断 2
			WR_2	O	外部总线的写信号线
			RxD_2	I	串口 1 的接收脚
			CMP-	I	比较器负极输入

编号			名称	类型	说明
LQFP44	PDIP40	LQFP32			
25	28	20	P3.7	I/O	标准 IO 口
			INT3	I	外部中断 3
			RD_2	O	外部总线的读信号线
			TxD_2	O	串口 1 的发送脚
			CMP+	I	比较器正极输入
26	29		P4.1	I/O	标准 IO 口
			ALE	O	地址锁存信号
			CMPO_2	O	比较器输出
27	30	21	P2.0	I/O	标准 IO 口
			A8	I	地址总线
			RSTSV	-	端口的初始电平可 ISP 下载时配置
28			P4.2	I/O	标准 IO 口
			RD_3	O	外部总线的读信号线
			TxD2_2	O	串口 2 的发送脚
29	31	22	P2.1	I/O	标准 IO 口
			A9	I	地址总线
30	32	23	P2.2	I/O	标准 IO 口
			A10	I	地址总线
			SS_2	I	SPI 的从机选择脚（主机为输出）
31	33	24	P2.3	I/O	标准 IO 口
			A11	I	地址总线
			MOSI_2	I/O	SPI 主机输出从机输入
32	34	25	P2.4	I/O	标准 IO 口
			A12	I	地址总线
			MISO_2	I/O	SPI 主机输入从机输出
			SDA_2	I/O	I2C 接口的数据线
33	35	26	P2.5	I/O	标准 IO 口
			A13	I	地址总线
			SCLK_2	I/O	SPI 的时钟脚
			SCL_2	I/O	I2C 的时钟线
34	36	27	P2.6	I/O	标准 IO 口
			A14	I	地址总线
35	37	28	P2.7	I/O	标准 IO 口
			A15	I	地址总线

编号			名称	类型	说明
LQFP44	PDIP40	LQFP32			
36	38	29	P0.0	I/O	标准 IO 口
			AD0	I	地址总线
37	39	30	P0.1	I/O	标准 IO 口
			AD1	I	地址总线
38	40	31	P0.2	I/O	标准 IO 口
			AD2	I	地址总线
39			P4.3	I/O	标准 IO 口
			WR	O	外部总线的写信号线
40	1	32	P0.3	I/O	标准 IO 口
			AD3	I	地址总线
41	2		P0.4	I/O	标准 IO 口
			AD4	I	地址总线
			T3	I	定时器 3 外部时钟输入
42	3		P0.5	I/O	标准 IO 口
			AD5	I	地址总线
			T3CLKO	O	定时器 3 时钟分频输出
43	4		P0.6	I/O	标准 IO 口
			AD6	I	地址总线
			T4	I	定时器 4 外部时钟输入
44	5		P0.7	I/O	标准 IO 口
			AD7	I	地址总线
			T4CLKO	O	定时器 4 时钟分频输出
1	6	1	P1.0	I/O	标准 IO 口

2.5 STC8F1K08S2-20Pin/16Pin系列

2.5.1 特性及价格

- 选型价格（不需要外部复位）

单片机型号	工作电压（V）	Flash 程序存储器 10 万次 字节	大容量扩展 SRAM 字节	强大的双 DPTR 可增可减	EEPROM 10 万次 字节	I/O 口最多数量	串口并可掉电唤醒	SPI	I²C	定时器/计数器（T0-T4 外部管脚也可掉电唤醒）	16 位高级 PWM 定时器 互补对称死区	15 位增强型 PWM（带死区控制）	PCA/CCP/PWM（可当外部中断并可掉电唤醒）	掉电唤醒专用定时器	15 路高速 ADC(8 路 PWM 可当 8 路 D/A 使用)	比较器（可当 1 路 A/D，可作外部掉电检测）	内部低压检测中断并可掉电唤醒	看门狗 复位定时器	内部高可靠复位（可选复位门槛电压）	内部高精准时钟（24MHz 可调）	可对外输出时钟及复位	程序加密后传输（防拦截）	可设置下次更新程序需口令	支持 RS485 下载	支持 USB 直接下载	本身就可在线仿真	封装				2018 年新品供货信息
																											TSSOP20	QFN20<3mm*3mm>	SOP16	SOP8	
STC8F1K08S2	2.0-5.5	8K	1.2K	2	3K	18	2	有	有	3	-	-	-	有		有	有	有	4 级	有	是	有	是	是	是	是	¥1.15	¥1.15	¥1.1	-	大量供货

注：以上的单价为 10K 及以上订货量的价格，量小则每片需增加 0.1 元人民币。当订货的总额达到或高于 3000 元时，可免运费发货，否则需要由客户承担运费。零售 10 片起售。

- **内核**
  - ✓ 超高速 8051 内核（1T），比传统 8051 约快 12 倍以上
  - ✓ 指令代码完全兼容传统 8051
  - ✓ 14 个中断源，4 级中断优先级
  - ✓ 支持在线仿真
- **工作电压**
  - ✓ 2.0V～5.5V
  - ✓ 内建 LDO
- **工作温度**
  - ✓ -40℃～85℃（超温度范围应用请参考电器特性章节说明）
- **Flash 存储器**
  - ✓ 最大 11K 字节 FLASH 程序存储器（ROM），用于存储用户代码
  - ✓ 支持用户配置 EEPROM 大小，512 字节单页擦除，擦写次数可达 10 万次以上
  - ✓ 支持在系统编程方式（ISP）更新用户应用程序，无需专用编程器
  - ✓ 支持单芯片仿真，无需专用仿真器，理论断点个数无限制
- **SRAM**
  - ✓ 128 字节内部直接访问 RAM（DATA）
  - ✓ 128 字节内部间接访问 RAM（IDATA）
  - ✓ 1024 字节内部扩展 RAM（内部 XDATA）
- **时钟控制**
  - ✓ 内部 24MHz 高精度 IRC（ISP 编程时可进行上下调整）

- 误差±0.3%（常温下 25℃）
- -1.8%～+0.8%温漂（全温度范围，-40℃～85℃）
- -1.0%～+0.5%温漂（温度范围，-20℃～65℃）

✓ 内部 32KHz 低速 IRC（误差较大）

✓ **注意：此型号不可使用外部晶振作为系统时钟源**

➢ **复位**

✓ 硬件复位
- 上电复位
- 复位脚复位（高电平复位），出厂时 P5.4 默认为 IO 口，ISP 下载时可将 P5.4 管脚设置为复位脚
- 看门狗溢出复位
- 低压检测复位，提供 4 级低压检测电压：2.2V、2.4V、2.7V、3.0V

✓ 软件复位
- 软件方式写复位触发寄存器

➢ **中断**

✓ 提供 14 个中断源：INT0、INT1、INT2、INT3、INT4、定时器 0、定时器 1、定时器 2、串口 1、串口 2、LVD 低压检测、SPI、I^2C、比较器

✓ 提供 4 级中断优先级

➢ **数字外设**

✓ 3 个 16 位定时器：定时器 0、定时器 1、定时器 2，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式

✓ 2 个高速串口：串口 1、串口 2，波特率时钟源最快可为 FOSC/4

✓ SPI：支持主机模式和从机模式以及主机/从机自动切换

✓ I^2C：支持主机模式和从机模式

➢ **模拟外设**

✓ 比较器，一组比较器

➢ **GPIO**

✓ 最多可达 18 个 GPIO： P1.0~P1.7、P3.0~P3.7、P5.4~P5.5

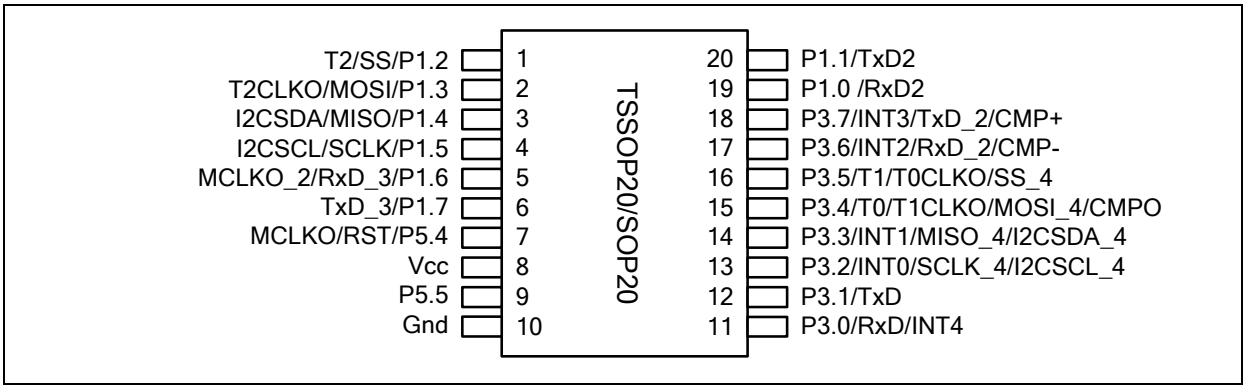
✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式

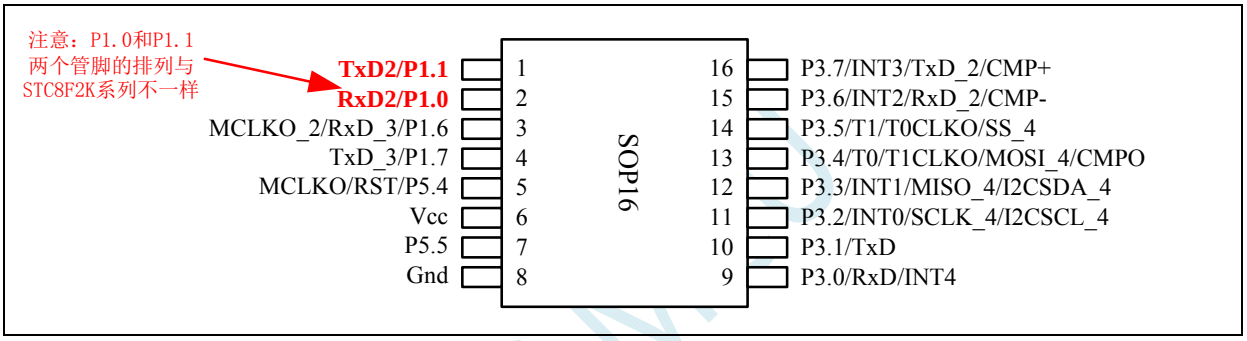
➢ **封装**

✓ TSSOP20、SOP16

STCMCU

2.5.2 管脚图


T2/SS/P1.2 1
T2CLKO/MOSI/P1.3 2
I2CSDA/MISO/P1.4 3
I2CSCL/SCLK/P1.5 4
MCLKO_2/RxD_3/P1.6 5
TxD_3/P1.7 6
MCLKO/RST/P5.4 7
Vcc 8
P5.5 9
Gnd 10
TSSOP20/SOP20
20 P1.1/TxD2
19 P1.0 /RxD2
18 P3.7/INT3/TxD_2/CMP+
17 P3.6/INT2/RxD_2/CMP-
16 P3.5/T1/T0CLKO/SS_4
15 P3.4/T0/T1CLKO/MOSI_4/CMPO
14 P3.3/INT1/MISO_4/I2CSDA_4
13 P3.2/INT0/SCLK_4/I2CSCL_4
12 P3.1/TxD
11 P3.0/RxD/INT4


注意：P1.0和P1.1
两个管脚的排列与
STC8F2K系列不一样
TxD2/P1.1 1
RxD2/P1.0 2
MCLKO_2/RxD_3/P1.6 3
TxD_3/P1.7 4
MCLKO/RST/P5.4 5
Vcc 6
P5.5 7
Gnd 8
SOP16
16 P3.7/INT3/TxD_2/CMP+
15 P3.6/INT2/RxD_2/CMP-
14 P3.5/T1/T0CLKO/SS_4
13 P3.4/T0/T1CLKO/MOSI_4/CMPO
12 P3.3/INT1/MISO_4/I2CSDA_4
11 P3.2/INT0/SCLK_4/I2CSCL_4
10 P3.1/TxD
9 P3.0/RxD/INT4

2.5.3 管脚说明

编号		名称	类型	说明
TSSOP20	SOP16			
1		P1.2	I/O	标准 IO 口
		SS	I	SPI 的从机选择脚（主机为输出）
		T2	I	定时器 2 外部时钟输入
2		P1.3	I/O	标准 IO 口
		MOSI	I/O	SPI 主机输出从机输入
		T2CLKO	O	定时器 2 时钟分频输出
3		P1.4	I/O	标准 IO 口
		MISO	I/O	SPI 主机输入从机输出
		SDA	I/O	I2C 接口的数据线
4		P1.5	I/O	标准 IO 口
		SCLK	I/O	SPI 的时钟脚
		SCL	I/O	I2C 的时钟线
5	3	P1.6	I/O	标准 IO 口
		RxD_3	I	串口 1 的接收脚
		MCLKO_2	O	主时钟分频输出
6	4	P1.7	I/O	标准 IO 口
		TxD_3	O	串口 1 的发送脚
7	5	P5.4	I/O	标准 IO 口
		RST	I	复位引脚
		MCLKO	O	主时钟分频输出
8	6	Vcc	VCC	电源脚
9	7	P5.5	I/O	标准 IO 口
10	8	Gnd	GND	地线
11	9	P3.0	I/O	标准 IO 口
		RxD	I	串口 1 的接收脚
		INT4	I	外部中断 4
12	10	P3.1	I/O	标准 IO 口
		TxD	O	串口 1 的发送脚

编号		名称	类型	说明
TSSOP20	SOP16			
13	11	P3.2	I/O	标准 IO 口
		INT0	I	外部中断 0
		SCL_4	I/O	I2C 的时钟线
		SCLK_4	I/O	SPI 的时钟脚
14	12	P3.3	I/O	标准 IO 口
		INT1	I	外部中断 1
		SDA_4	I/O	I2C 接口的数据线
		MISO_4	I/O	SPI 主机输入从机输出
15	13	P3.4	I/O	标准 IO 口
		T0	I	定时器 0 外部时钟输入
		T1CLKO	O	定时器 1 时钟分频输出
		MOSI_4	I/O	SPI 主机输出从机输入
		CMPO	O	比较器输出
16	14	P3.5	I/O	标准 IO 口
		T1	I	定时器 1 外部时钟输入
		T0CLKO	O	定时器 0 时钟分频输出
		SS_4	I	SPI 的从机选择脚（主机为输出）
17	15	P3.6	I/O	标准 IO 口
		INT2	I	外部中断 2
		RxD_2	I	串口 1 的接收脚
		CMP-	I	比较器负极输入
18	16	P3.7	I/O	标准 IO 口
		INT3	I	外部中断 3
		TxD_2	O	串口 1 的发送脚
		CMP+	I	比较器正极输入
19	2	P1.0	I/O	标准 IO 口
		RxD2	I	串口 2 的接收脚
20	1	P1.1	I/O	标准 IO 口
		TxD2	O	串口 2 的发送脚

2.6 STC8F1K08-8Pin系列

2.6.1 特性及价格

- 选型价格（不需要外部复位）

单片机型号	工作电压（V）	Flash 程序存储器 10 万次 字节	大容量扩展 SRAM 字节	强大的双 DPTR 可增可减	EEPROM 10 万次 字节	I/O 口最多数量	串口并可掉电唤醒	SPI	I²C	定时器/计数器（T0-T4 外部管脚也可掉电唤醒）	16 位高级 PWM 定时器 互补对称死区	15 位增强型 PWM（带死区控制）	PCA/CCP/PWM（可当外部中断并可掉电唤醒）	掉电唤醒专用定时器	15 路高速 ADC(8 路 PWM 可当 8 路 D/A 使用)	比较器（可当 1 路 A/D，可作外部掉电检测）	内部低压检测中断并可掉电唤醒	看门狗 复位定时器	内部高可靠复位（可选复位门槛电压）	内部高精准时钟（24MHz 可调）	可对外输出时钟及复位	程序加密后传输（防拦截）	可设置下次更新程序需口令	支持 RS485 下载	支持 USB 直接下载	本身就可在线仿真	封装				2018 年新品供货信息
																											TSSOP20	QFN20<3mm*3mm>	SOP16	SOP8	
STC8F1K08	2.0-5.5	8K	1.2K	2	3K	6	1	有	有	3	-	-	-	有		–	有	有	4 级	有	是	有	是	是	是	是	-	-	-	¥1.05	大量供货

注：以上的单价为 10K 及以上订货量的价格，量小则每片需增加 0.1 元人民币。当订货的总额达到或高于 3000 元时，可免运费发货，否则需要由客户承担运费。零售 10 片起售。

- **内核**
  - ✓ 超高速 8051 内核（1T），比传统 8051 约快 12 倍以上
  - ✓ 指令代码完全兼容传统 8051
  - ✓ 13 个中断源，4 级中断优先级
  - ✓ 支持在线仿真
- **工作电压**
  - ✓ 2.0V～5.5V
  - ✓ 内建 LDO
- **工作温度**
  - ✓ -40℃～85℃（超温度范围应用请参考电器特性章节说明）
- **Flash 存储器**
  - ✓ 最大 11K 字节 FLASH 程序存储器（ROM），用于存储用户代码
  - ✓ 支持用户配置 EEPROM 大小，512 字节单页擦除，擦写次数可达 10 万次以上
  - ✓ 支持在系统编程方式（ISP）更新用户应用程序，无需专用编程器
  - ✓ 支持单芯片仿真，无需专用仿真器，理论断点个数无限制
- **SRAM**
  - ✓ 128 字节内部直接访问 RAM（DATA）
  - ✓ 128 字节内部间接访问 RAM（IDATA）
  - ✓ 1024 字节内部扩展 RAM（内部 XDATA）
- **时钟控制**
  - ✓ 内部 24MHz 高精度 IRC（ISP 编程时可进行上下调整）

- 误差±0.3%（常温下 25℃）
- -1.8%～+0.8%温漂（全温度范围，-40℃～85℃）
- -1.0%～+0.5%温漂（温度范围，-20℃～65℃）

✓ 内部 32KHz 低速 IRC（误差较大）

✓ **注意：此型号不可使用外部晶振作为系统时钟源**

➢ **复位**

✓ 硬件复位
- 上电复位
- 复位脚复位（高电平复位），出厂时 P5.4 默认为 IO 口，ISP 下载时可将 P5.4 管脚设置为复位脚
- 看门狗溢出复位
- 低压检测复位，提供 4 级低压检测电压：2.2V、2.4V、2.7V、3.0V

✓ 软件复位
- 软件方式写复位触发寄存器

➢ **中断**

✓ 提供 13 个中断源：INT0、INT1、INT2、INT3、INT4、定时器 0、定时器 1、定时器 2、串口 1、LVD 低压检测、SPI、I^2C、比较器

✓ 提供 4 级中断优先级

➢ **数字外设**

✓ 3 个 16 位定时器：定时器 0、定时器 1、定时器 2，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式

✓ 1 个高速串口：串口 1，波特率时钟源最快可为 FOSC/4

✓ SPI：支持主机模式和从机模式以及主机/从机自动切换

✓ I^2C：支持主机模式和从机模式

➢ **GPIO**

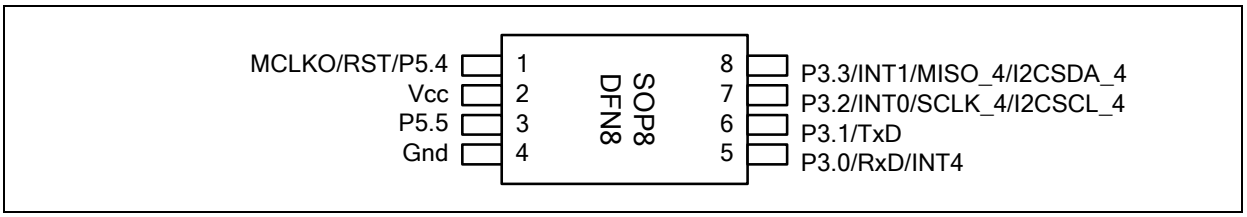
✓ 最多可达 6 个 GPIO： P3.0~P3.3、P5.4~P5.5

✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式

➢ **封装**

✓ SOP8

2.6.2 管脚图



2.6.3 管脚说明

编号 SOP8	名称	类型	说明
1	P5.4	I/O	标准 IO 口
	RST	I	复位引脚
	MCLKO	O	主时钟分频输出
2	Vcc	VCC	电源脚
3	P5.5	I/O	标准 IO 口
4	Gnd	GND	地线
5	P3.0	I/O	标准 IO 口
	RxD	I	串口 1 的接收脚
	INT4	I	外部中断 4
6	P3.1	I/O	标准 IO 口
	TxD	O	串口 1 的发送脚
7	P3.2	I/O	标准 IO 口
	INT0	I	外部中断 0
	SCL_4	I/O	I2C 的时钟线
	SCLK_4	I/O	SPI 的时钟脚
8	P3.3	I/O	标准 IO 口
	INT1	I	外部中断 1
	SDA_4	I/O	I2C 接口的数据线
	MISO_4	I/O	SPI 主机输入从机输出

3 功能脚切换

STC8A/STC8F 系列单片机的特殊外设串口 1、串口 2、串口 3、串口 4、SPI、PCA、PWM、I^2C 以及总线控制脚可以在多个 I/O 直接进行切换，以实现一个外设当作多个设备进行分时复用。

3.1 功能脚切换相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
BUS_SPEED	总线速度控制寄存器	A1H	RW_S[1:0]						SPEED[1:0]		00xx,xx00
P_SW1	外设端口切换寄存器 1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-	nn00,000x
P_SW2	外设端口切换寄存器 2	BAH	EAXFR	~~CAN_S~~	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S	0x00,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWM0CR	PWM0 控制寄存器	FF04H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI	00x0,0000
PWM1CR	PWM1 控制寄存器	FF14H	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI	00x0,0000
PWM2CR	PWM2 控制寄存器	FF24H	ENC2O	C2INI	-	C2_S[1:0]		EC2I	EC2T2SI	EC2T1SI	00x0,0000
PWM3CR	PWM3 控制寄存器	FF34H	ENC3O	C3INI	-	C3_S[1:0]		EC3I	EC3T2SI	EC3T1SI	00x0,0000
PWM4CR	PWM4 控制寄存器	FF44H	ENC4O	C4INI	-	C4_S[1:0]		EC4I	EC4T2SI	EC4T1SI	00x0,0000
PWM5CR	PWM5 控制寄存器	FF54H	ENC5O	C5INI	-	C5_S[1:0]		EC5I	EC5T2SI	EC5T1SI	00x0,0000
PWM6CR	PWM6 控制寄存器	FF64H	ENC6O	C6INI	-	C6_S[1:0]		EC6I	EC6T2SI	EC6T1SI	00x0,0000
PWM7CR	PWM7 控制寄存器	FF74H	ENC7O	C7INI	-	C7_S[1:0]		EC7I	EC7T2SI	EC7T1SI	00x0,0000
CKSEL	时钟选择寄存器	FE00H	MCLKODIV[3:0]				MCLKO_S	-	MCKSEL[1:0]		0000,0000

3.1.1 总线速度控制寄存器（BUS_SPEED）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
BUS_SPEED	A1H	RW_S[1:0]						SPEED[1:0]	

RW_S[1:0]：外部总线 RD/WR 控制线选择位

RW_S[1:0]	RD	WR
00	P4.4	P4.3
01	P3.7	P3.6
10	P4.2	P4.0
11	保留	

3.1.2 外设端口切换控制寄存器 1（P_SW1），串口 1、CCP、SPI 切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-

S1_S[1:0]：串口 1 功能脚选择位

S1_S[1:0]	RxD	TxD
00	P3.0	P3.1
01	P3.6	P3.7
10	P1.6	P1.7
11	P4.3	P4.4

CCP_S[1:0]：PCA 功能脚选择位

CCP_S[1:0]	ECI	CCP0	CCP1	CCP2	CCP3
00	P1.2	P1.7	P1.6	P1.5	P1.4
01	P2.2	P2.3	P2.4	P2.5	P2.6
10	P7.4	P7.0	P7.1	P7.2	P7.3
11	P3.5	P3.3	P3.2	P3.1	P3.0

SPI_S[1:0]：SPI 功能脚选择位

SPI_S[1:0]	SS	MOSI	MISO	SCLK
00	P1.2	P1.3	P1.4	P1.5
01	P2.2	P2.3	P2.4	P2.5
10	P7.4	P7.5	P7.6	P7.7
11	P3.5	P3.4	P3.3	P3.2

3.1.3 外设端口切换控制寄存器 2（P_SW2），串口 2/3/4、I2C、比较器输出切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S

EAXFR：扩展 RAM 区特殊功能寄存器（XFR）访问控制寄存器

0：禁止访问 XFR

1：使能访问 XFR。

当需要访问 XFR 时，必须先将 EAXFR 置 1，才能对 XFR 进行正常的读写

I2C_S[1:0]：I^2C 功能脚选择位

I2C_S[1:0]	SCL	SDA
00	P1.5	P1.4
01	P2.5	P2.4
10	P7.7	P7.6
11	P3.2	P3.3

CMPO_S：比较器输出脚选择位

CMPO_S	CMPO

0	P3.4
1	P4.1

S4_S：串口 4 功能脚选择位

S4_S	RxD4	TxD4
0	P0.2	P0.3
1	P5.2	P5.3

S3_S：串口 3 功能脚选择位

S3_S	RxD3	TxD3
0	P0.0	P0.1
1	P5.0	P5.1

S2_S：串口 2 功能脚选择位

S2_S	RxD2	TxD2
0	P1.0	P1.1
1	P4.0	P4.2

3.1.4 时钟选择寄存器（CKSEL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CKSEL	FE00H	MCLKODIV[3:0]				MCLKO_S	-	MCKSEL[1:0]	

MCLKO_S：主时钟输出脚选择位

MCLKO_S	MCLKO
0	P5.4
1	P1.6

3.1.5 增强型PWM控制寄存器（PWMnCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0CR	FF04H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI
PWM1CR	FF14H	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI
PWM2CR	FF24H	ENC2O	C2INI	-	C2_S[1:0]		EC2I	EC2T2SI	EC2T1SI
PWM3CR	FF34H	ENC3O	C3INI	-	C3_S[1:0]		EC3I	EC3T2SI	EC3T1SI
PWM4CR	FF44H	ENC4O	C4INI	-	C4_S[1:0]		EC4I	EC4T2SI	EC4T1SI
PWM5CR	FF54H	ENC5O	C5INI	-	C5_S[1:0]		EC5I	EC5T2SI	EC5T1SI
PWM6CR	FF64H	ENC6O	C6INI	-	C6_S[1:0]		EC6I	EC6T2SI	EC6T1SI
PWM7CR	FF74H	ENC7O	C7INI	-	C7_S[1:0]		EC7I	EC7T2SI	EC7T1SI

C0_S[1:0]：增强型 PWM 通道 0 输出脚选择位

C0_S[1:0]	PWM0
00	P2.0
01	P1.0
10	P6.0
11	保留

C1_S[1:0]：增强型 PWM 通道 1 输出脚选择位

C1_S[1:0]	PWM1
00	P2.1

01	P1.1
10	P6.1
11	保留

C2_S[1:0]：增强型 PWM 通道 2 输出脚选择位

C2_S[1:0]	PWM2
00	P2.2
01	P1.2
10	P6.2
11	保留

C3_S[1:0]：增强型 PWM 通道 3 输出脚选择位

C3_S[1:0]	PWM3
00	P2.3
01	P1.3
10	P6.3
11	保留

C4_S[1:0]：增强型 PWM 通道 4 输出脚选择位

C4_S[1:0]	PWM4
00	P2.4
01	P1.4
10	P6.4
11	保留

C5_S[1:0]：增强型 PWM 通道 5 输出脚选择位

C5_S[1:0]	PWM5
00	P2.5
01	P1.5
10	P6.5
11	保留

C6_S[1:0]：增强型 PWM 通道 6 输出脚选择位

C6_S[1:0]	PWM6
00	P2.6
01	P1.6
10	P6.6
11	保留

C7_S[1:0]：增强型 PWM 通道 7 输出脚选择位

C7_S[1:0]	PWM7
00	P2.7
01	P1.7
10	P6.7
11	保留

STC MCU

3.2 范例程序

3.2.1 串口 1 切换

汇编代码

```
P_SW1   DATA    0A2H

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

        MOV     P_SW1,#00H          ;RXD/P3.0, TXD/P3.1
;       MOV     P_SW1,#40H          ;RXD_2/P3.6, TXD_2/P3.7
;       MOV     P_SW1,#80H          ;RXD_3/P1.6, TXD_3/P1.7
;       MOV     P_SW1,#0C0H         ;RXD_4/P4.3, TXD_4/P4.4

        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

sfr P_SW1 = 0xa2;

void main()
{
        P_SW1 = 0x00;               //RXD/P3.0, TXD/P3.1
//      P_SW1 = 0x40;               //RXD_2/P3.6, TXD_2/P3.7
//      P_SW1 = 0x80;               //RXD_3/P1.6, TXD_3/P1.7
//      P_SW1 = 0xc0;               //RXD_4/P4.3, TXD_4/P4.4

        while (1);
}
```

3.2.2 串口 2 切换

汇编代码

```
P_SW2   DATA    0BAH

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

        MOV     P_SW2,#00H          ;RXD2/P1.0, TXD2/P1.1
;       MOV     P_SW2,#01H          ;RXD2_2/P4.0, TXD2_2/P4.2

        SJMP    $
```

```
        END
```

C 语言代码

```
#include "reg51.h"

sfr P_SW2 = 0xba;

void main()
{
        P_SW2 = 0x00;               //RXD2/P1.0, TXD2/P1.1
//      P_SW2 = 0x01;               //RXD2_2/P4.0, TXD2_2/P4.2

        while (1);
}
```

3.2.3 串口 3 切换

汇编代码

```
P_SW2   DATA    0BAH

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

        MOV     P_SW2,#00H              ;RXD3/P0.0, TXD3/P0.1
;       MOV     P_SW2,#02H              ;RXD3_2/P5.0, TXD3_2/P5.1

        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

sfr P_SW2 = 0xba;

void main()
{
        P_SW2 = 0x00;               // RXD3/P0.0, TXD3/P0.1
//      P_SW2 = 0x02;               // RXD3_2/P5.0, TXD3_2/P5.1

        while (1);
}
```

3.2.4 串口 4 切换

汇编代码

```
P_SW2   DATA    0BAH

        ORG     0000H
```

```
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

        MOV     P_SW2,#00H              ;RXD4/P0.2, TXD4/P0.3
;       MOV     P_SW2,#04H              ;RXD4_2/P5.2, TXD4_2/P5.3

        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

sfr P_SW2 = 0xba;

void main()
{
        P_SW2 = 0x00;           //RXD4/P0.2, TXD4/P0.3
//      P_SW2 = 0x04;           //RXD4_2/P5.2, TXD4_2/P5.3

        while (1);
}
```

3.2.5 SPI切换

汇编代码

```
P_SW1   DATA    0A2H

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

        MOV     P_SW1,#00H              ;SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5
;       MOV     P_SW1,#04H              ;SS_2/P2.2, MOSI_2/P2.3, MISO_2/P2.4, SCLK_2/P2.5
;       MOV     P_SW1,#08H              ;SS_3/P7.4, MOSI_3/P7.5, MISO_3/P7.6, SCLK_3/P7.7
;       MOV     P_SW1,#0CH              ;SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2

        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

sfr P_SW1 = 0xa2;

void main()
{
        P_SW1 = 0x00;           //SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5
//      P_SW1 = 0x04;           //SS_2/P2.2, MOSI_2/P2.3, MISO_2/P2.4, SCLK_2/P2.5
```

```
//      P_SW1 = 0x08;           //SS_3/P7.4, MOSI_3/P7.5, MISO_3/P7.6, SCLK_3/P7.7
//      P_SW1 = 0x0c;           //SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2

        while (1);
}
```

3.2.6 PWM切换

汇编代码

```
P_SW2       DATA    0BAH
PWM0CR      EQU     0FF04H
PWM1CR      EQU     0FF14H
PWM2CR      EQU     0FF24H
PWM3CR      EQU     0FF34H
PWM4CR      EQU     0FF44H
PWM5CR      EQU     0FF54H
PWM6CR      EQU     0FF64H
PWM7CR      EQU     0FF74H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #3FH

            MOV     P_SW2,#80H
            MOV     A,#00H          ;PWM0/P2.0
;           MOV     A,#08H          ;PWM0_2/P1.0
;           MOV     A,#10H          ;PWM0_3/P6.0
            MOV     DPTR,#PWM0CR
            MOVX    @DPTR,A
            MOV     A,#00H          ;PWM1/P2.1
;           MOV     A,#08H          ;PWM1_2/P1.1
;           MOV     A,#10H          ;PWM1_3/P6.1
            MOV     DPTR,#PWM1CR
            MOVX    @DPTR,A
            MOV     A,#00H          ;PWM2/P2.2
;           MOV     A,#08H          ;PWM2_2/P1.2
;           MOV     A,#10H          ;PWM2_3/P6.2
            MOV     DPTR,#PWM2CR
            MOVX    @DPTR,A
            MOV     A,#00H          ;PWM3/P2.3
;           MOV     A,#08H          ;PWM3_2/P1.3
;           MOV     A,#10H          ;PWM3_3/P6.3
            MOV     DPTR,#PWM3CR
            MOVX    @DPTR,A
            MOV     A,#00H          ;PWM4/P2.4
;           MOV     A,#08H          ;PWM4_2/P1.4
;           MOV     A,#10H          ;PWM4_3/P6.4
            MOV     DPTR,#PWM4CR
            MOVX    @DPTR,A
            MOV     A,#00H          ;PWM5/P2.5
;           MOV     A,#08H          ;PWM5_2/P1.5
;           MOV     A,#10H          ;PWM5_3/P6.5
            MOV     DPTR,#PWM5CR
            MOVX    @DPTR,A
            MOV     A,#00H          ;PWM6/P2.6
```

STC MCU

```
;               MOV     A,#08H                  ;PWM6_2/P1.6
;               MOV     A,#10H                  ;PWM6_3/P6.6
                MOV     DPTR,#PWM6CR
                MOVX    @DPTR,A
                MOV     A,#00H                  ;PWM7/P2.7
;               MOV     A,#08H                  ;PWM7_2/P1.7
;               MOV     A,#10H                  ;PWM7_3/P6.7
                MOV     DPTR,#PWM7CR
                MOVX    @DPTR,A
                MOV     P_SW2,#00H

                SJMP    $

                END
```

C 语言代码

```
#include "reg51.h"

#define PWM0CR      (*(unsigned char volatile xdata *)0xff04)
#define PWM1CR      (*(unsigned char volatile xdata *)0xff14)
#define PWM2CR      (*(unsigned char volatile xdata *)0xff24)
#define PWM3CR      (*(unsigned char volatile xdata *)0xff34)
#define PWM4CR      (*(unsigned char volatile xdata *)0xff44)
#define PWM5CR      (*(unsigned char volatile xdata *)0xff54)
#define PWM6CR      (*(unsigned char volatile xdata *)0xff64)
#define PWM7CR      (*(unsigned char volatile xdata *)0xff74)

sfr P_SW2 = 0xba;

void main()
{
        P_SW2 = 0x80;
        PWM0CR = 0x00;          //PWM0/P2.0
//      PWM0CR = 0x08;          //PWM0_2/P1.0
//      PWM0CR = 0x10;          //PWM0_3/P6.0
        PWM1CR = 0x00;          //PWM1/P2.1
//      PWM1CR = 0x08;          //PWM1_2/P1.1
//      PWM1CR = 0x10;          //PWM1_3/P6.1
        PWM2CR = 0x00;          //PWM2/P2.2
//      PWM2CR = 0x08;          //PWM2_2/P1.2
//      PWM2CR = 0x10;          //PWM2_3/P6.2
        PWM3CR = 0x00;          //PWM3/P2.3
//      PWM3CR = 0x08;          //PWM3_2/P1.3
//      PWM3CR = 0x10;          //PWM3_3/P6.3
        PWM4CR = 0x00;          //PWM4/P2.4
//      PWM4CR = 0x08;          //PWM4_2/P1.4
//      PWM4CR = 0x10;          //PWM4_3/P6.4
        PWM5CR = 0x00;          //PWM5/P2.5
//      PWM5CR = 0x08;          //PWM5_2/P1.5
//      PWM5CR = 0x10;          //PWM5_3/P6.5
        PWM6CR = 0x00;          //PWM6/P2.6
//      PWM6CR = 0x08;          //PWM6_2/P1.6
//      PWM6CR = 0x10;          //PWM6_3/P6.6
        PWM7CR = 0x00;          //PWM7/P2.7
//      PWM7CR = 0x08;          //PWM7_2/P1.7
//      PWM7CR = 0x10;          //PWM7_3/P6.7
        P_SW2 = 0x00;
```

```
        while (1);
}
```

3.2.7 PCA/CCP/PWM切换

汇编代码

```
P_SW1   DATA    0A2H

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

        MOV     P_SW1,#00H          ;ECI/P1.2, CCP0/P1.7, CCP1/P1.6, CCP2/P1.5,CCP3/P1.4
;       MOV     P_SW1,#10H          ;ECI_2/P2.2, CCP0_2/P2.3, CCP1_2/P2.4, CCP2_2/P2.5,CCP3_2/P2.6
;       MOV     P_SW1,#20H          ;ECI_3/P7.4, CCP0_3/P7.0, CCP1_3/P7.1, CCP2_3/P7.2,CCP3_3/P7.3
;       MOV     P_SW1,#30H          ;ECI_4/P3.5, CCP0_4/P3.3, CCP1_4/P3.2, CCP2_4/P3.1,CCP3_4/P3.0

        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

sfr P_SW1 = 0xa2;

void main()
{
        P_SW1 = 0x00;               //ECI/P1.2, CCP0/P1.7, CCP1/P1.6, CCP2/P1.5,CCP3/P1.4
//      P_SW1 = 0x10;               //ECI_2/P2.2, CCP0_2/P2.3, CCP1_2/P2.4, CCP2_2/P2.5,CCP3_2/P2.6
//      P_SW1 = 0x20;               //ECI_3/P7.4, CCP0_3/P7.0, CCP1_3/P7.1, CCP2_3/P7.2,CCP3_3/P7.3
//      P_SW1 = 0x30;               //ECI_4/P3.5, CCP0_4/P3.3, CCP1_4/P3.2, CCP2_4/P3.1,CCP3_4/P3.0

        while (1);
}
```

3.2.8 I2C切换

汇编代码

```
P_SW2   DATA    0BAH

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

        MOV     P_SW2,#00H          ;SCL/P1.5, SDA/P1.4
;       MOV     P_SW2,#10H          ;SCL_2/P2.5, SDA_2/P2.4
;       MOV     P_SW2,#20H          ;SCL_3/P7.7, SDA_3/P7.6
;       MOV     P_SW2,#30H          ;SCL_4/P3.2, SDA_4/P3.3
```

```
        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

sfr P_SW2 = 0xba;

void main()
{
        P_SW2 = 0x00;           //SCL/P1.5, SDA/P1.4
//      P_SW2 = 0x10;           //SCL_2/P2.5, SDA_2/P2.4
//      P_SW2 = 0x20;           //SCL_3/P7.7, SDA_3/P7.6
//      P_SW2 = 0x30;           //SCL_4/P3.2, SDA_4/P3.3

        while (1);
}
```

3.2.9 比较器输出切换

汇编代码

```
P_SW2   DATA    0BAH

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

        MOV     P_SW2,#00H          ;CMPO/P3.4
;       MOV     P_SW2,#08H          ;CMPO_2/P4.1

        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

sfr P_SW2 = 0xba;

void main()
{
        P_SW2 = 0x00;           //CMPO/P3.4
//      P_SW2 = 0x08;           //CMPO_2/P4.1

        while (1);
}
```

STC MCU

3.2.10 主时钟输出切换

汇编代码

```
P_SW2   DATA    0BAH
CKSEL   EQU     0FE00H

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

        MOV     P_SW2,#80H
        MOV     A,#40H                  ;IRC24M/4 output via MCLKO/P5.4
;       MOV     A,#48H                  ;IRC24M/4 output via MCLKO_2/P1.6
;       MOV     A,#0E8H                 ;IRC24M/128 output via MCLKO_2/P1.6
        MOV     DPTR,#CKSEL
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

#define CKSEL     (*(unsigned char volatile xdata *)0xfe00)

sfr P_SW2 = 0xba;

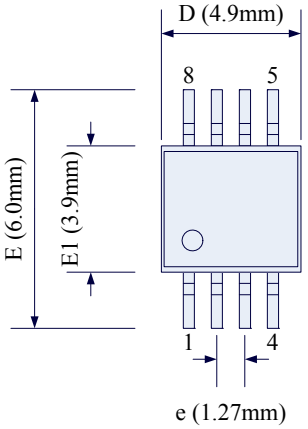
void main()
{
        P_SW2 = 0x80;
        CKSEL = 0x40;               //IRC24M/4 output via MCLKO/P5.4
//      CKSEL = 0x48;               //IRC24M/4 output via MCLKO_2/P1.6
//      CKSEL = 0xe8;               //IRC24M/128 output via MCLKO_2/P1.6
        P_SW2 = 0x00;

        while (1);
}
```

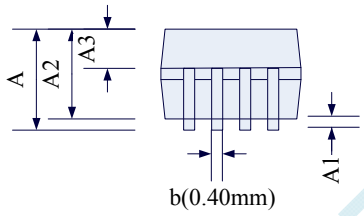
STC MCU

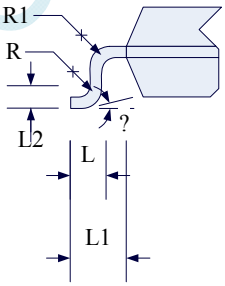
4 封装尺寸图

4.1 SOP8 封装尺寸图



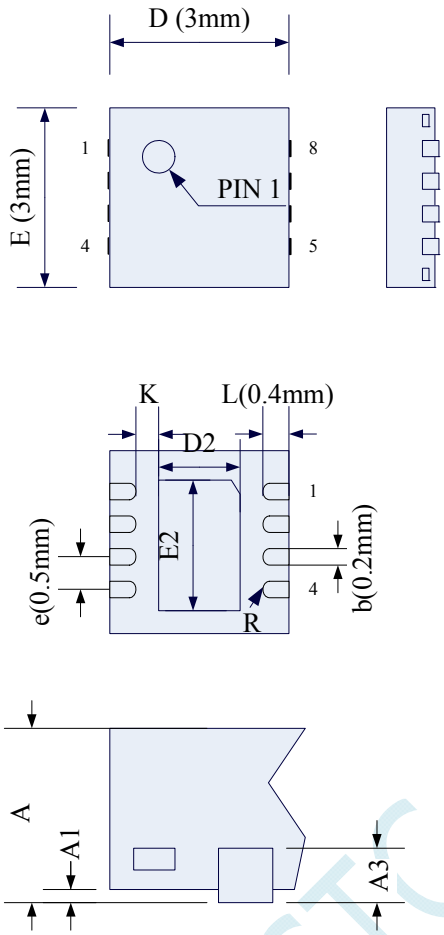
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	1.35	1.60	1.75
A1	0.10	0.15	0.25
A2	1.25	1.45	1.65
A3	0.55	0.65	0.75
b	0.35	0.40	0.45
D	4.80	4.90	5.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e		1.27BSC	
L	0.45	0.60	0.80
L1		1.04REF	
L2		0.25BSC	
R1	0.07	-	-
R2	0.07	-	-





STC MCU

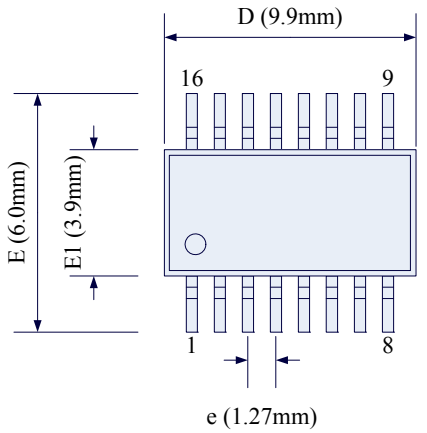
4.2 DFN8 封装尺寸图（3mm*3mm）



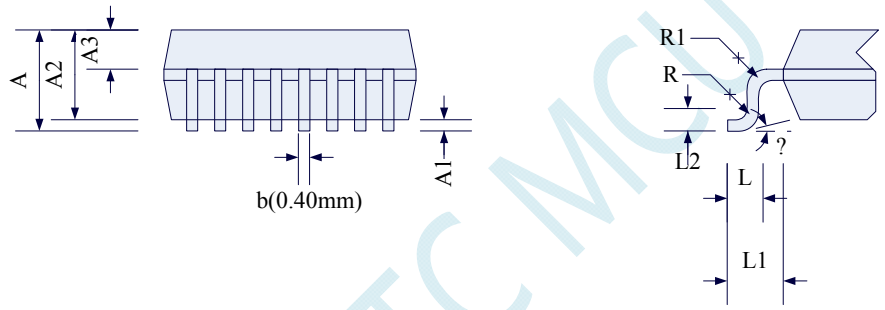
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
A2	0.50	0.55	0.60
A3	-	0.203REF	-
b	0.175	0.20	0.225
D	2.89	3.00	3.11
E	2.89	3.00	3.11
D2	1.65	1.70	1.75
E2	2.35	2.40	2.45
e	0.45	0.50	0.55
L	0.381	0.40	0.419
K	0.275REF		
R	0.125	-	-

STC 现有 DFN8 封装芯片的背面金属片（衬底），在芯片内部并未接地，在用户的 PCB 板上可以接地，也可以不接地，不会对芯片性能造成影响

4.3 SOP16 封装尺寸图

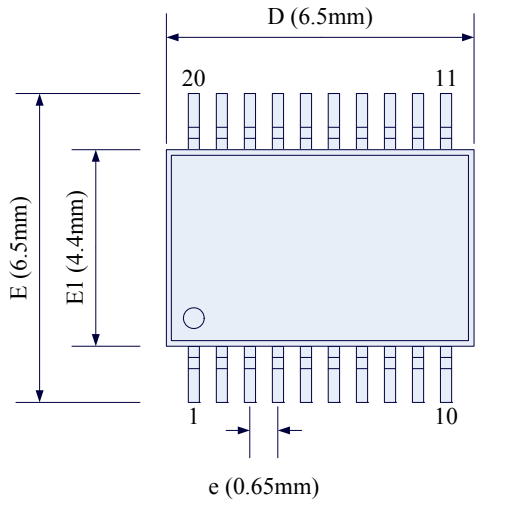


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	1.35	1.60	1.75
A1	0.10	0.15	0.25
A2	1.25	1.45	1.65
A3	0.55	0.65	0.75
b	0.35	0.40	0.45
D	9.80	9.90	10.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e		1.27BSC	
L	0.45	0.60	0.80
L1		1.04REF	
L2		0.25BSC	
R1	0.07	-	-
R2	0.07	-	-

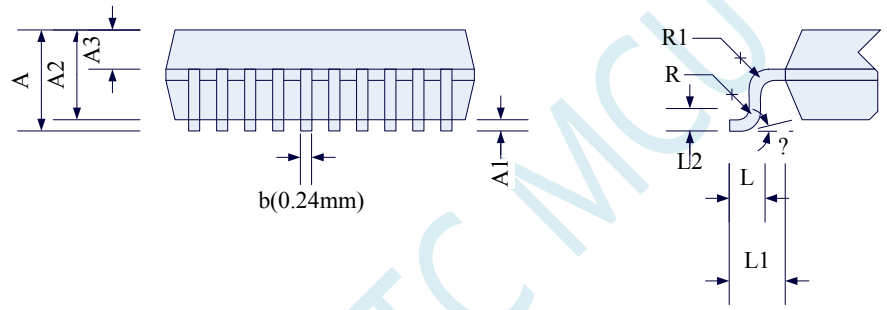


STC MCU

4.4 TSSOP20 封装尺寸图

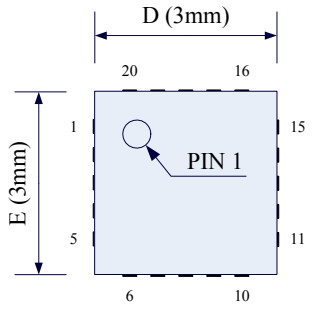


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.20
A1	0.05	-	0.15
A2	0.90	1.00	1.05
A3	0.34	0.44	0.54
b	0.20	0.24	0.28
D	6.40	6.50	6.60
E	6.20	6.50	6.60
E1	4.30	4.40	4.50
e		0.65BSC	
L	0.45	0.60	0.75
L1		1.00REF	
L2		0.25BSC	
R1	0.09	-	-
R2	0.09	-	-

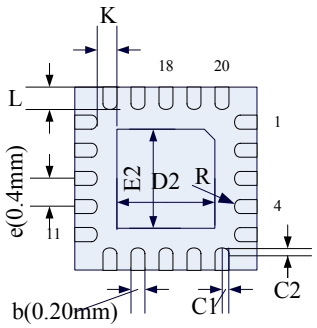


STC MCU

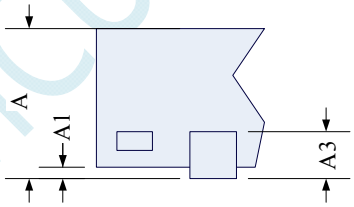
4.5 QFN20 封装尺寸图（3mm*3mm）





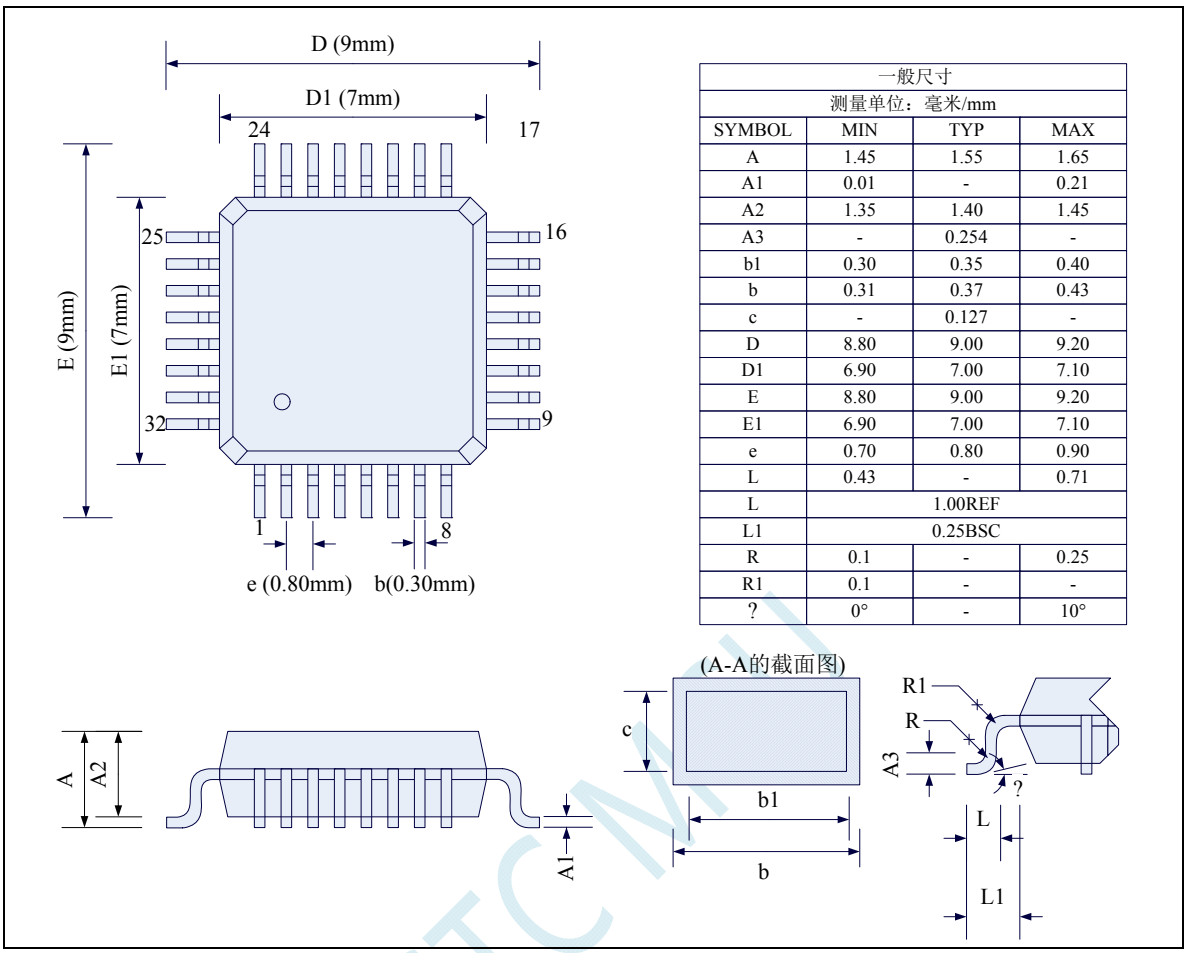


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
A2	0.50	0.55	0.60
A3	-	0.20REF	-
b	0.15	0.20	0.25
D	2.90	3.00	3.10
E	2.90	3.00	3.10
D2	1.40	1.50	1.60
E2	1.40	1.50	1.60
e	0.30	0.40	0.50
L	0.35	0.40	0.45
K		0.35REF	
R	0.085	-	-
C1	-	0.07	-
C2	-	0.07	-



STC 现有 QFN20 封装芯片的背面金属片（衬底），在芯片内部并未接地，在用户的 PCB 板上可以接地，也可以不接地，不会对芯片性能造成影响

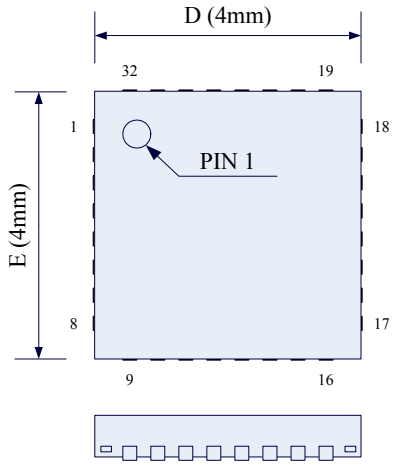
4.6 LQFP32 封装尺寸图（9mm*9mm）

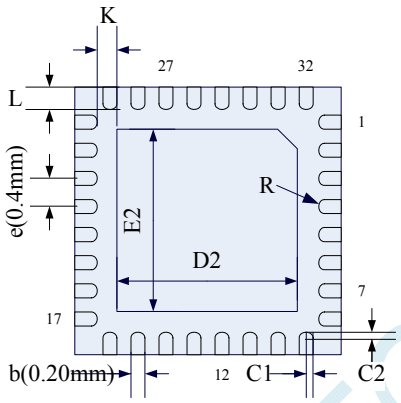


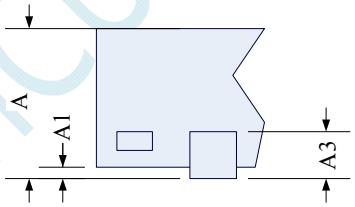
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	1.45	1.55	1.65
A1	0.01	-	0.21
A2	1.35	1.40	1.45
A3	-	0.254	-
b1	0.30	0.35	0.40
b	0.31	0.37	0.43
c	-	0.127	-
D	8.80	9.00	9.20
D1	6.90	7.00	7.10
E	8.80	9.00	9.20
E1	6.90	7.00	7.10
e	0.70	0.80	0.90
L	0.43	-	0.71
L	1.00REF		
L1	0.25BSC		
R	0.1	-	0.25
R1	0.1	-	-
?	0°	-	10°

STCMCU

4.7 QFN32 封装尺寸图（4mm*4mm）



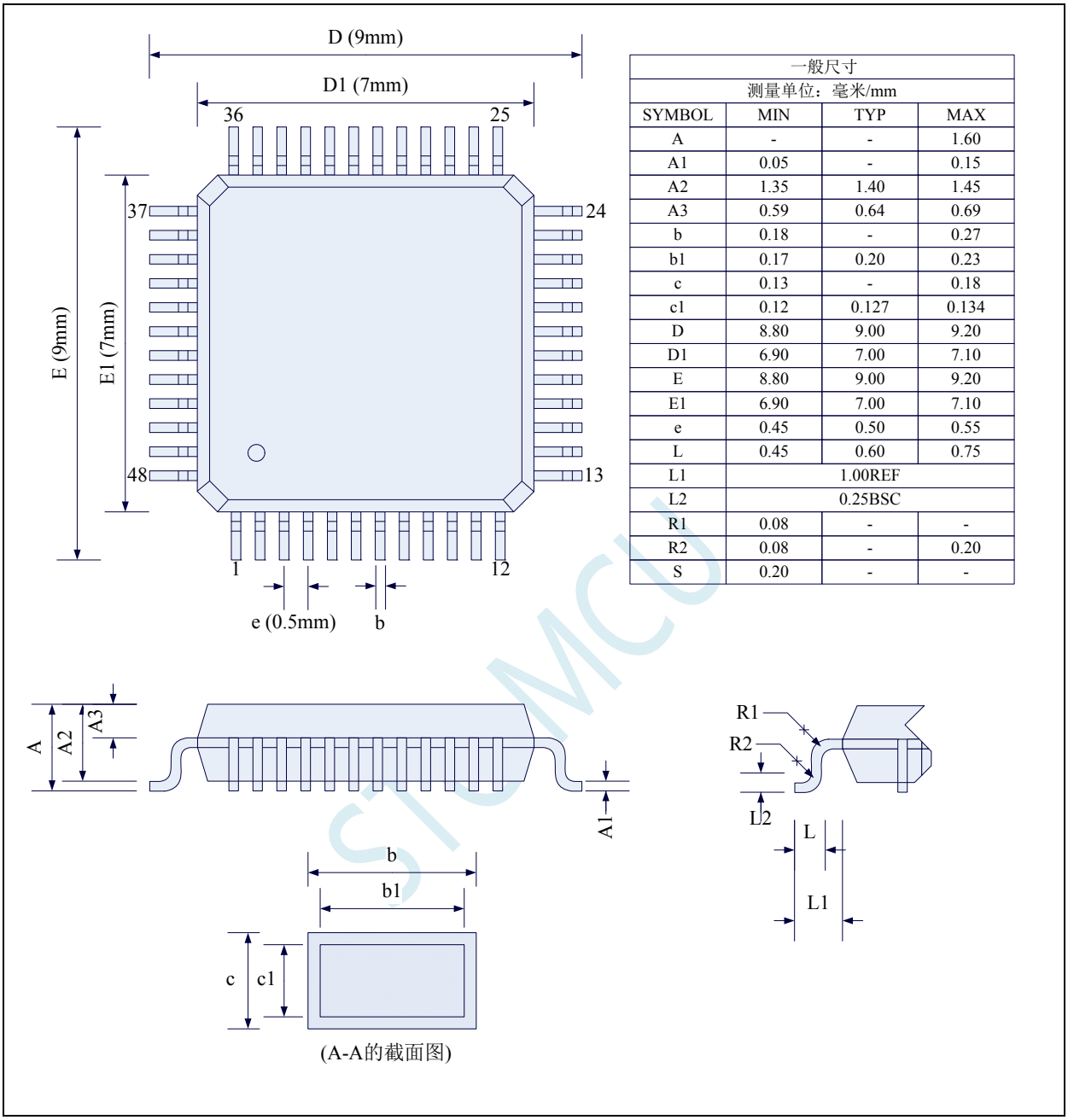




一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
A2	0.50	0.55	0.60
A3	-	0.20REF	-
b	0.15	0.20	0.25
D	3.90	4.00	4.10
E	3.90	4.00	4.10
D2	2.60	2.70	2.80
E2	2.60	2.70	2.80
e	0.30	0.40	0.50
L	0.35	0.40	0.45
K		0.25REF	
R	0.09	-	-
C1	-	0.16	-
C2	-	0.16	-

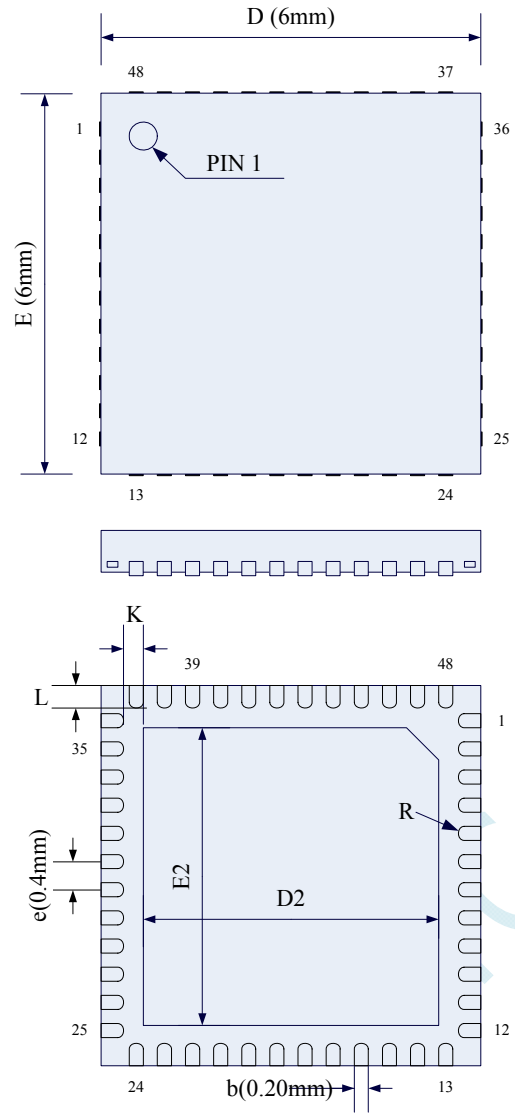
STC 现有 QFN32 封装芯片的背面金属片（衬底），在芯片内部并未接地，在用户的 PCB 板上可以接地，也可以不接地，不会对芯片性能造成影响

4.8 LQFP48 封装尺寸图（9mm*9mm）



一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	-	0.27
b1	0.17	0.20	0.23
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	8.80	9.00	9.20
D1	6.90	7.00	7.10
E	8.80	9.00	9.20
E1	6.90	7.00	7.10
e	0.45	0.50	0.55
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-

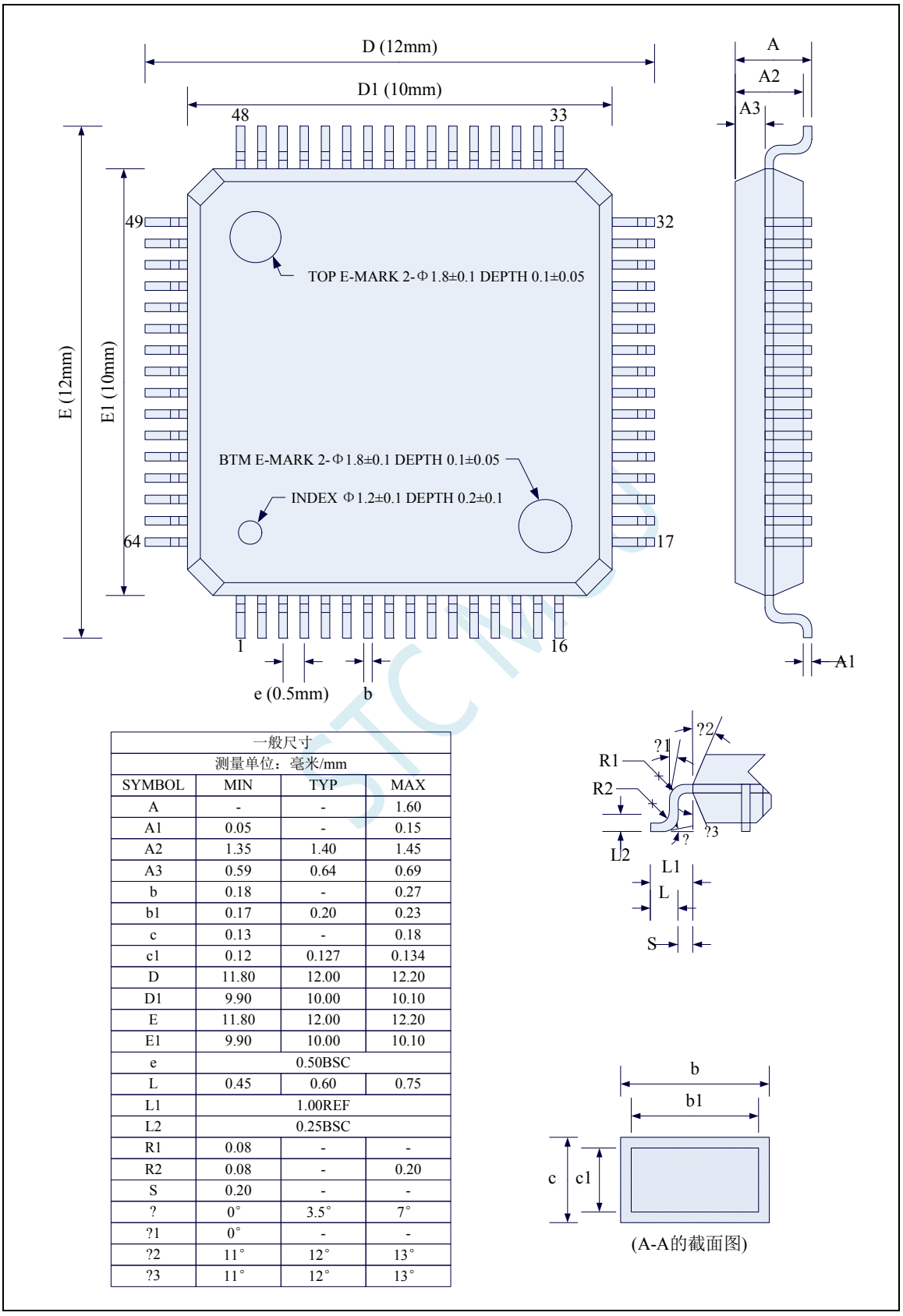
4.9 QFN48 封装尺寸图（6mm*6mm）



一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A3		0.20REF	
b	0.15	0.20	0.25
D	5.90	6.00	6.10
E	5.90	6.00	6.10
D2	3.95	4.05	4.15
E2	3.95	4.05	4.15
e	0.35	0.40	0.45
L	0.35	0.40	0.45
K		0.20	
R	0.09	-	-

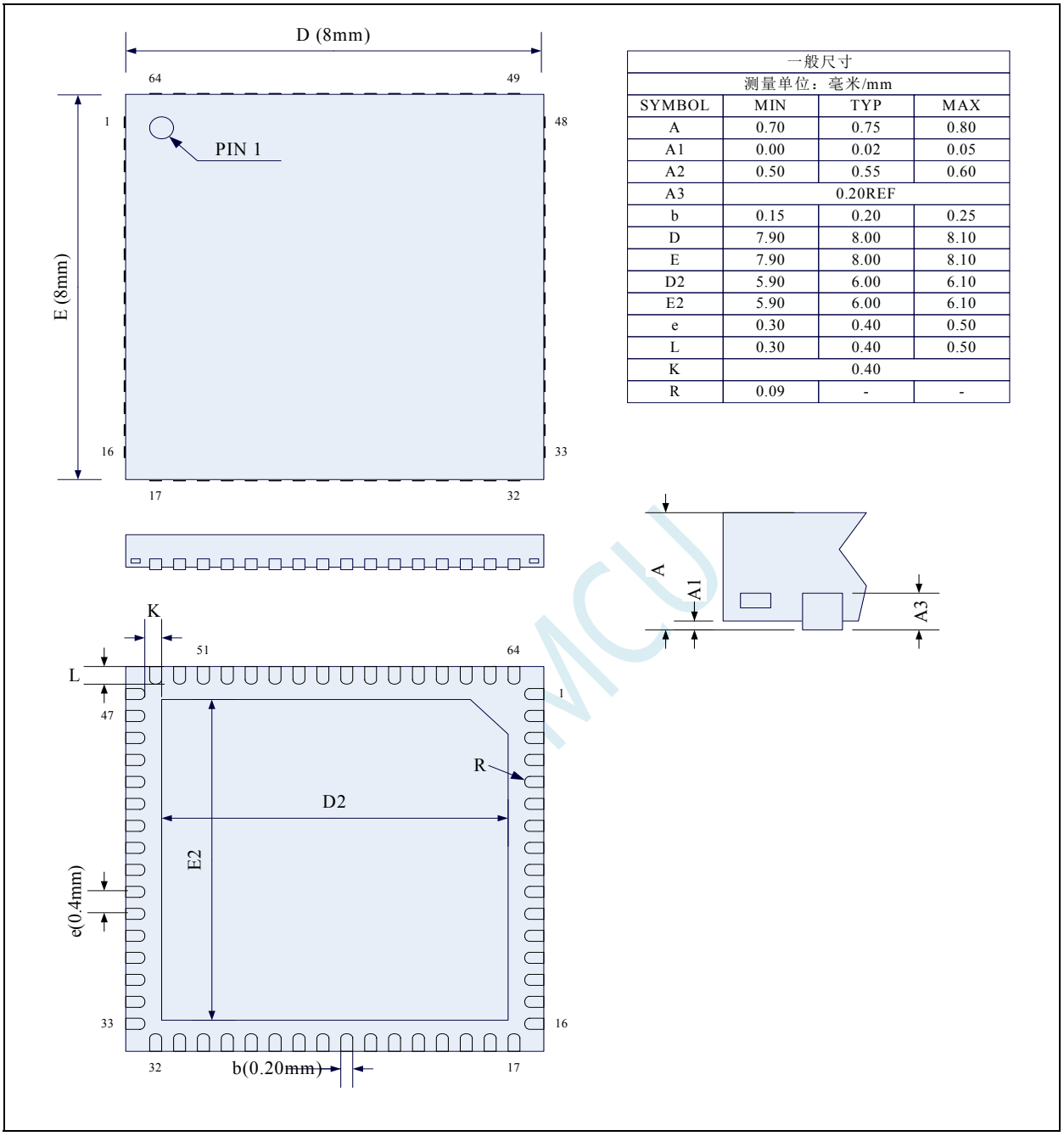
STC 现有 QFN48 封装芯片的背面金属片（衬底），在芯片内部并未接地，在用户的 PCB 板上可以接地，也可以不接地，不会对芯片性能造成影响

4.10 LQFP64S封装尺寸图（12mm*12mm）



一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	-	0.27
b1	0.17	0.20	0.23
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	11.80	12.00	12.20
D1	9.90	10.00	10.10
E	11.80	12.00	12.20
E1	9.90	10.00	10.10
e	0.50BSC		
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-
?	0°	3.5°	7°
?1	0°	-	-
?2	11°	12°	13°
?3	11°	12°	13°

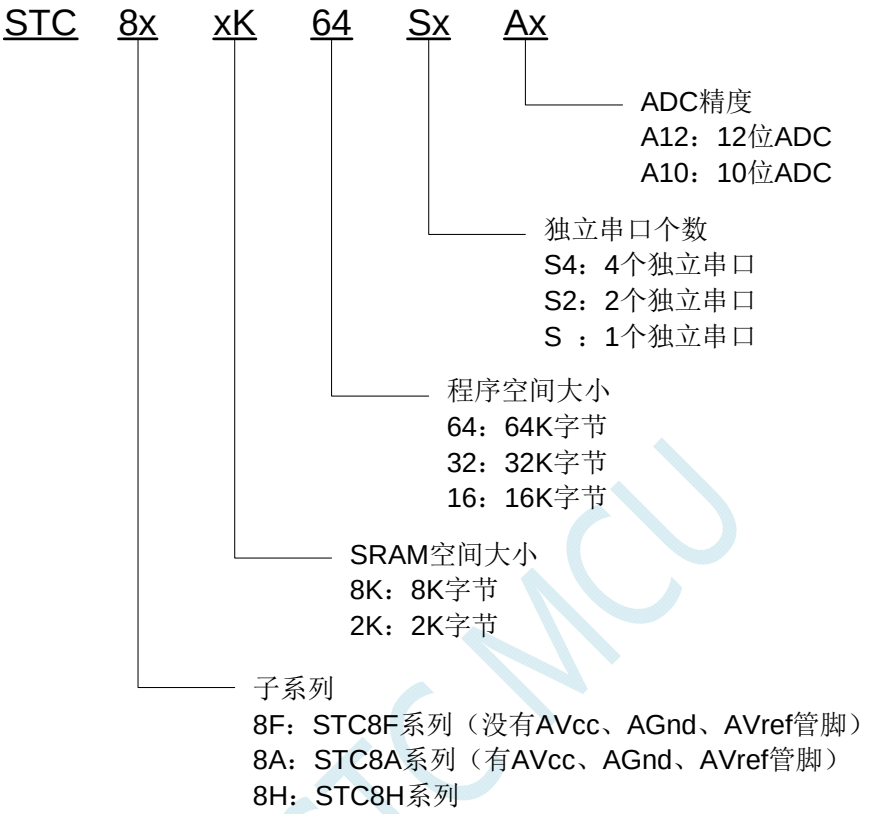
4.11 QFN64 封装尺寸图（8mm*8mm）



一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A2	0.50	0.55	0.60
A3		0.20REF	
b	0.15	0.20	0.25
D	7.90	8.00	8.10
E	7.90	8.00	8.10
D2	5.90	6.00	6.10
E2	5.90	6.00	6.10
e	0.30	0.40	0.50
L	0.30	0.40	0.50
K		0.40	
R	0.09	-	-

STC 现有 QFN64 封装芯片的背面金属片（衬底），在芯片内部并未接地，在用户的 PCB 板上可以接地，也可以不接地，不会对芯片性能造成影响

4.12 STC8A/8F系列单片机命名规则

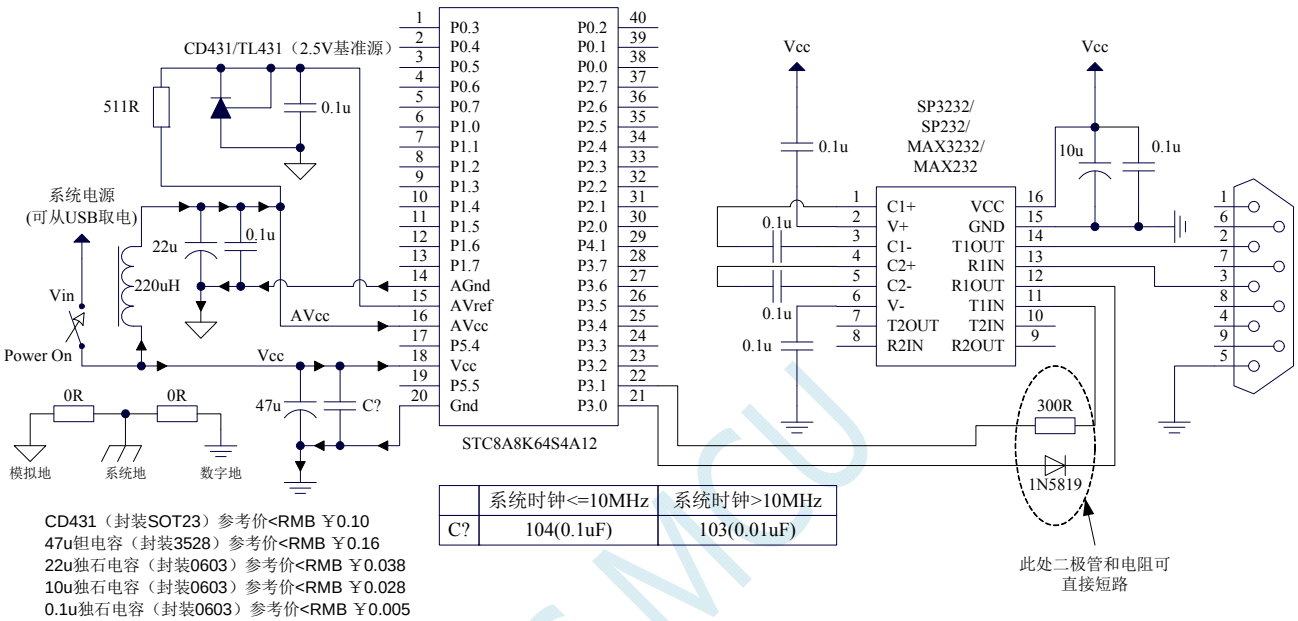


STC MCU

5 ISP下载及典型应用线路图

5.1 STC8A系列ISP下载应用线路图

5.1.1 使用RS-232 转换器下载，也可支持仿真（使用高精度ADC）



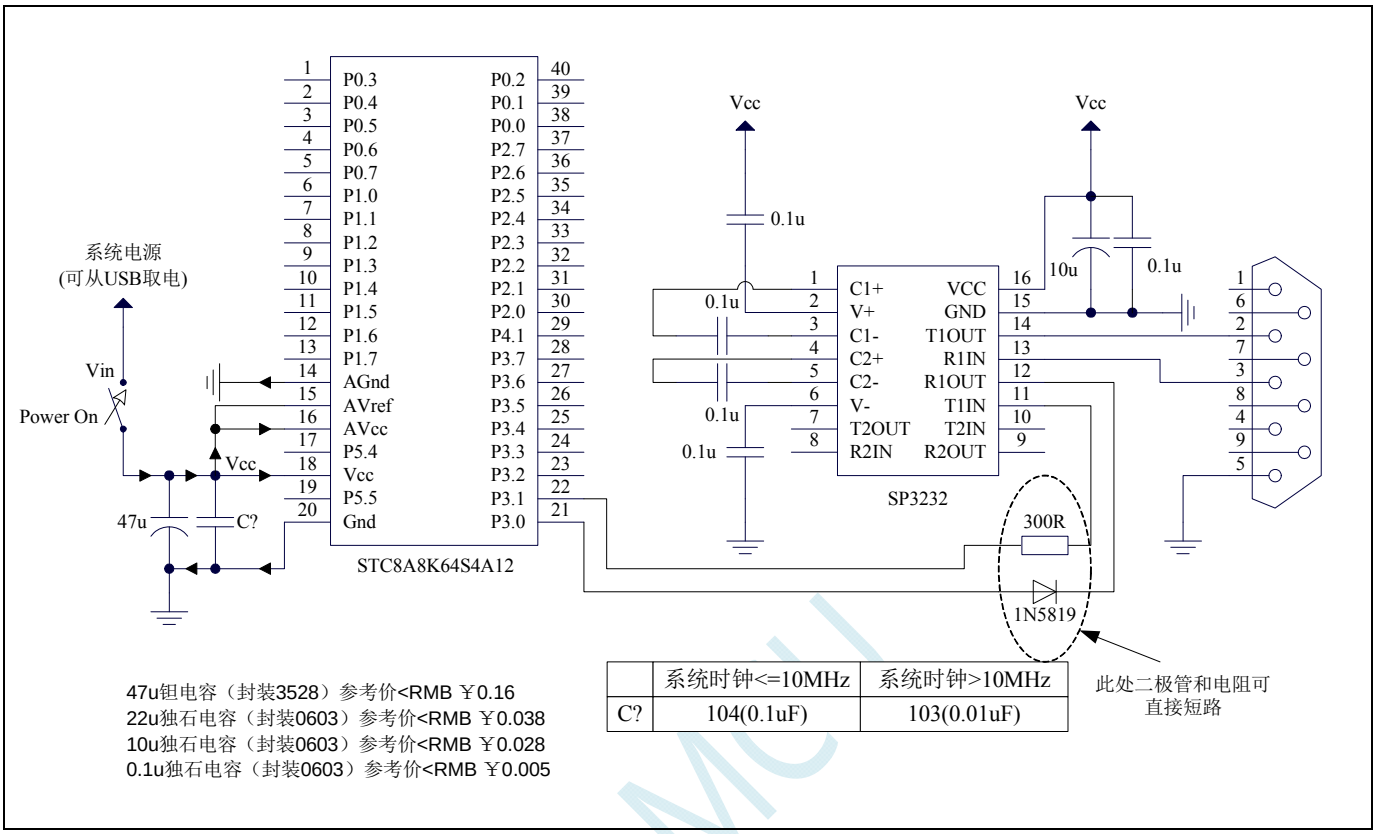
	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

CD431（封装SOT23）参考价<RMB ￥0.10
47u钽电容（封装3528）参考价<RMB ￥0.16
22u独石电容（封装0603）参考价<RMB ￥0.038
10u独石电容（封装0603）参考价<RMB ￥0.028
0.1u独石电容（封装0603）参考价<RMB ￥0.005

ISP 下载步骤：

1、 给目标芯片停电
2、 点击 STC-ISP 下载软件中的“下载/编程”按钮
3、 给目标芯片上电
4、 开始 ISP 下载

5.1.2 使用RS-232 转换器下载，也可支持仿真（使用一般精度ADC）



47u钽电容（封装3528）参考价<RMB ￥0.16
22u独石电容（封装0603）参考价<RMB ￥0.038
10u独石电容（封装0603）参考价<RMB ￥0.028
0.1u独石电容（封装0603）参考价<RMB ￥0.005

	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

ISP 下载步骤：

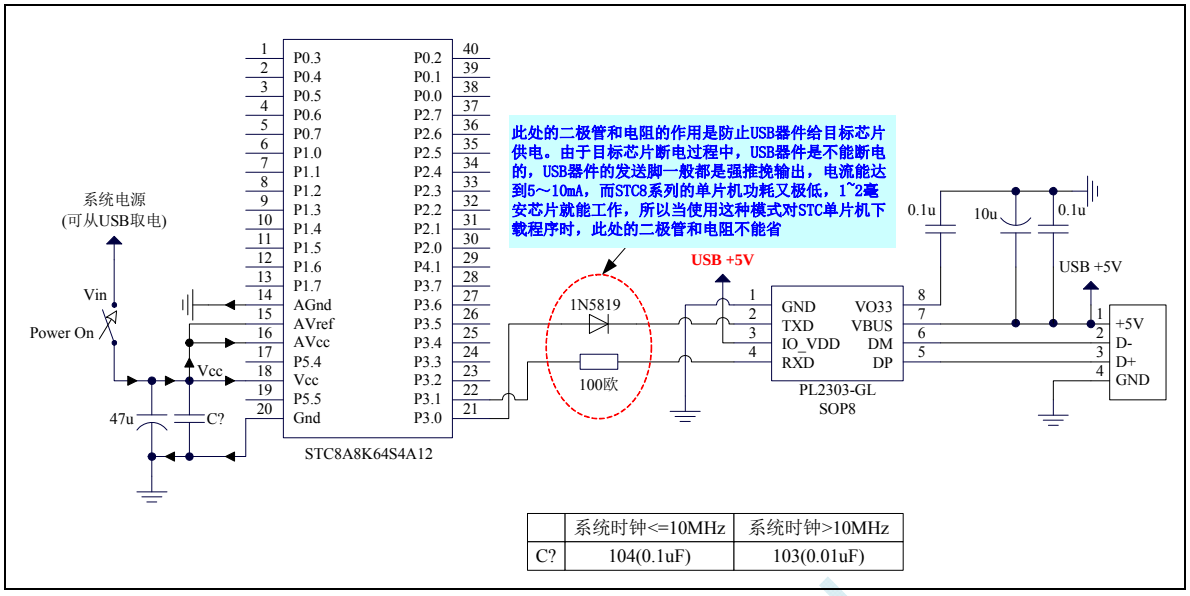
5、 给目标芯片停电

6、 点击 STC-ISP 下载软件中的“下载/编程”按钮

7、 给目标芯片上电

8、 开始 ISP 下载

5.1.3 使用PL2303-GL下载，也可支持仿真



	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

ISP 下载步骤：

1、 给目标芯片停电，注意不能给 USB 转串口芯片停电（如：CH340、PL2303-GL 等）

2、 由于 USB 转串口芯片的发送脚一般都是强推挽输出，必须在目标芯片的 P3.0 口和 USB 转串口芯片的发送脚之间串接一个二极管，否则目标芯片无法完全断电，达不到给目标芯片停电的目标。

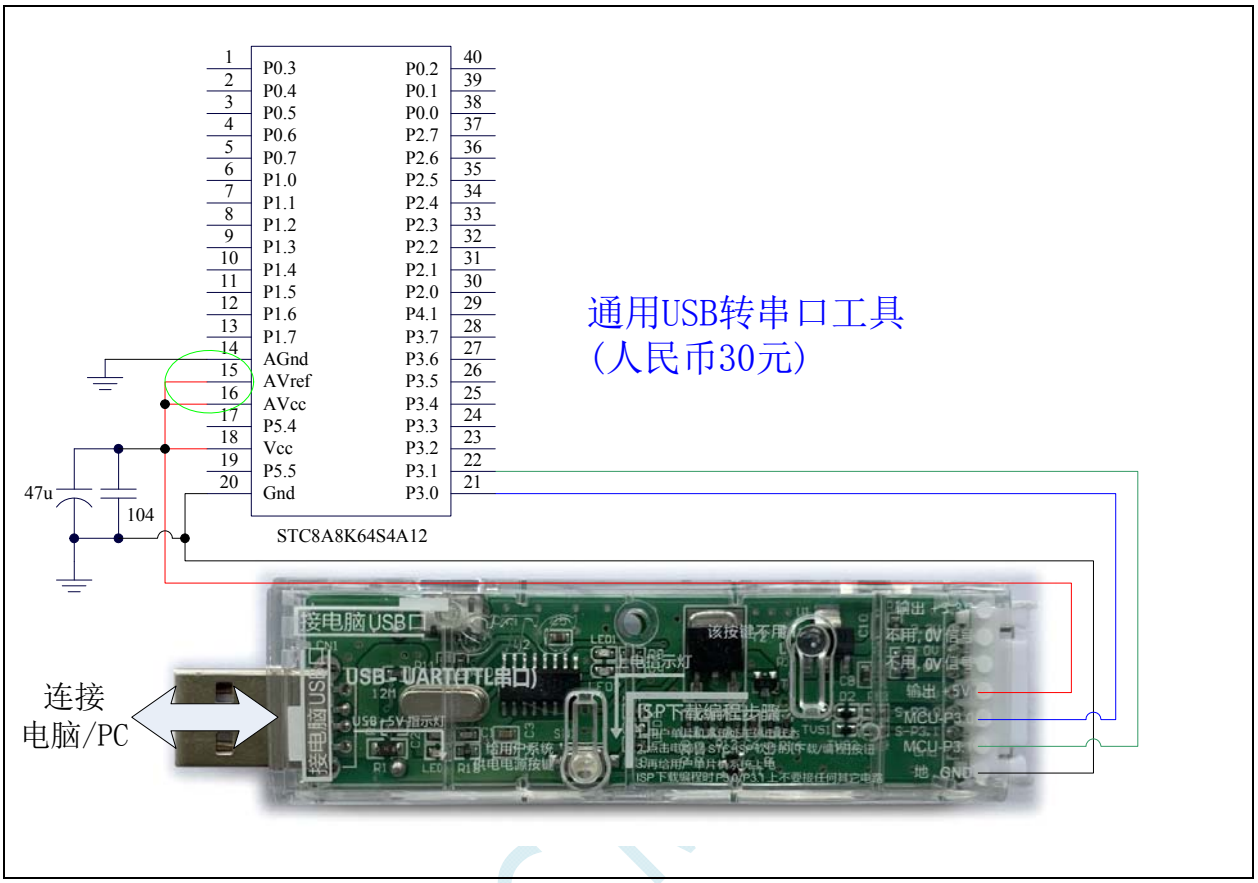
3、 点击 STC-ISP 下载软件中的“下载/编程”按钮

4、 给目标芯片上电

5、 开始 ISP 下载

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

5.1.4 使用通用USB转串口工具下载，支持ISP在线下载，也可支持仿真



ISP 下载步骤：

1、 按照如图所示的连接方式将通用 USB 转串口工具和目标芯片连接

2、 按下电源按钮，确定目标芯片处于**停电状态**（上电指示灯为灭的状态）。**注意：工具第一次上电时是不对外供电的，因此若是第一次上电使用此工具，可跳过此步。**

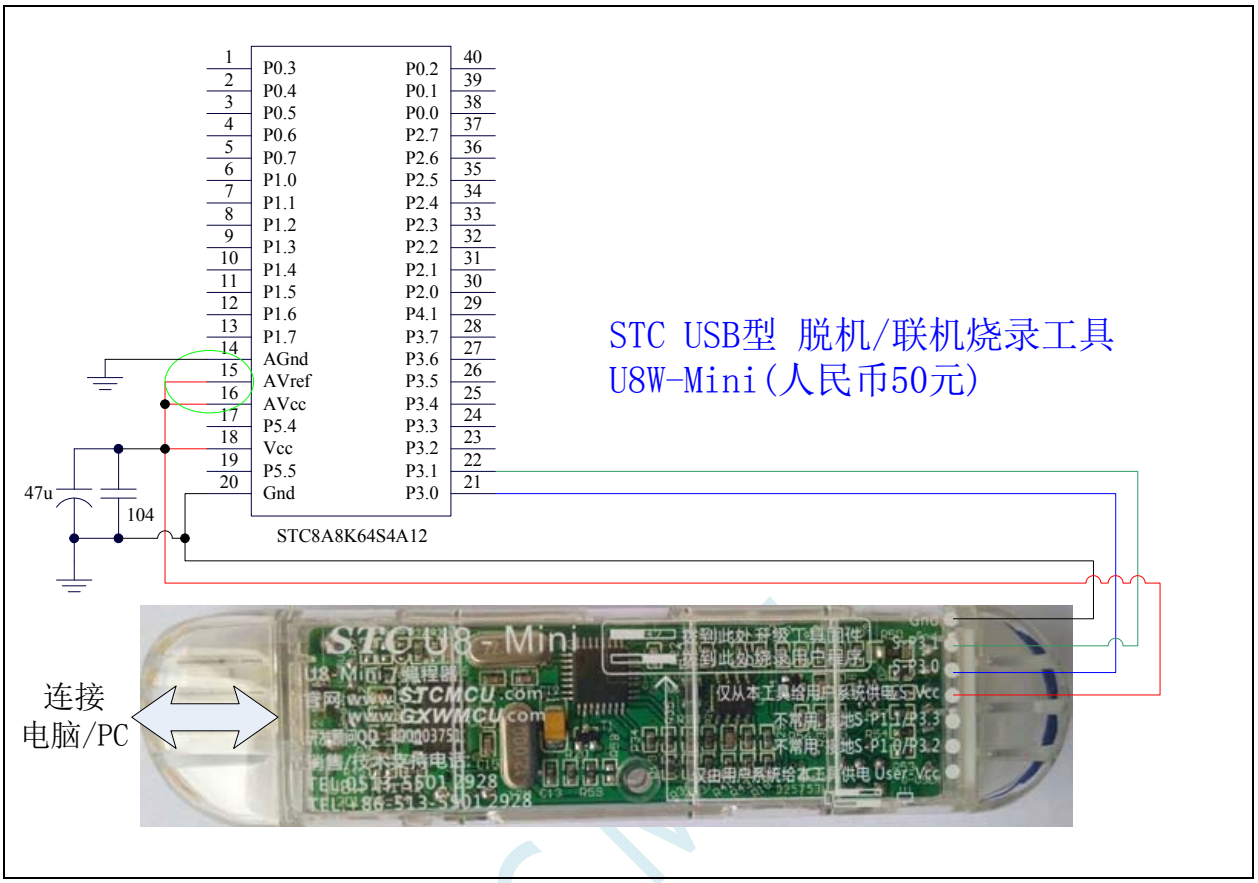
3、 点击 STC-ISP 下载软件中的“下载/编程”按钮

4、 再次按下电源按钮，给目标芯片上电（上电指示灯为亮的状态）

5、 开始 ISP 下载

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

5.1.5 使用U8-Mini工具下载，支持ISP在线和脱机下载，也可支持仿真



ISP 下载步骤：

1、 按照如图所示的连接方式将 U8-Mini 和目标芯片连接

2、 点击 STC-ISP 下载软件中的“下载/编程”按钮

3、 开始 ISP 下载

注意：若是使用 U8-Mini 给目标系统供电，目标系统的总电流不能大于 200mA，否则会导致下载失败。

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

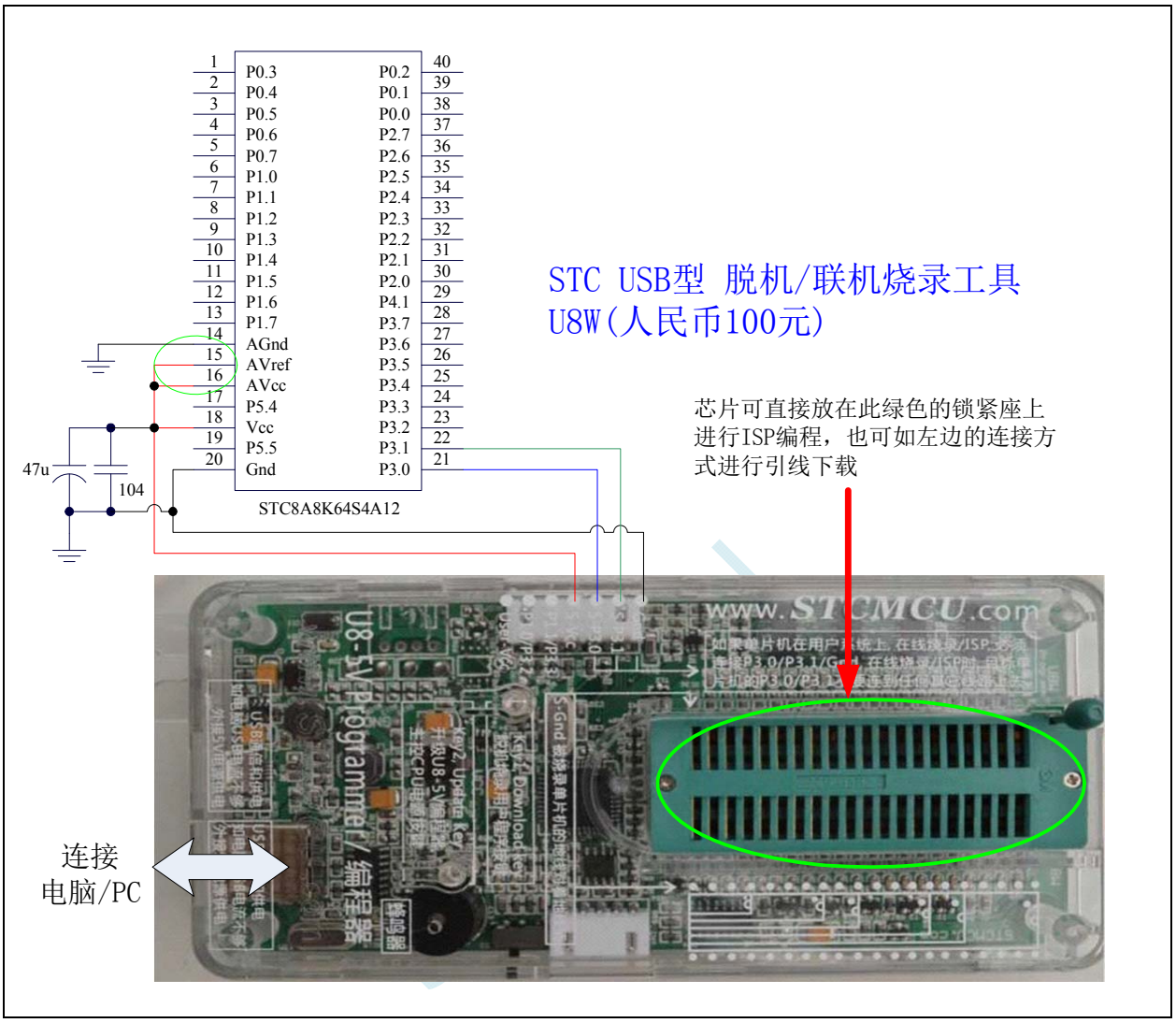
若要使用 U8-Mini 进行仿真，首先必须将 U8-Mini 设置为直通模式。U8W/U8W-Mini 实现 USB 转串口直通模式的方法如下：

1、 首先 U8W/U8W-Mini 固件必须升级到 v1.37 及以上版本

2、 U8W/U8W-Mini 上电后为正常下载模式，此时按住工具上的 Key1（下载）按键不要松开，再按一下 Key2（电源）按键，然后放开 Key2（电源）按键 后，再松开 Key1（下载）按键，U8W/U8W-Mini 会进入 USB 转串口直通模式。（**按下 Key1 ➔ 按下 Key2 ➔ 松开 Key2 ➔ 松开 Key1**）

3、 进入直通模式的 U8W/U8W-Mini 工具只是简单的 USB 转串口不具备脱机下载功能，若需要恢复 U8W/U8W-Mini 的原有功能，只需要再次单独按一下 Key2（电源）按键 即可

5.1.6 使用U8W工具下载，支持ISP在线和脱机下载，也可支持仿真



ISP 下载步骤（连线方式）：

1、 按照如图所示的连接方式将 U8W 和目标芯片连接

2、 点击 STC-ISP 下载软件中的“下载/编程”按钮

3、 开始 ISP 下载

注意：若是使用 U8W 给目标系统供电，目标系统的总电流不能大于 200mA，否则会导致下载失败。

ISP 下载步骤（在板方式）：

1、 将芯片按照 1 脚靠近锁紧扳手、管脚向下靠齐的方向放置好目标芯片

2、 点击 STC-ISP 下载软件中的“下载/编程”按钮

3、 开始 ISP 下载

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

若要使用 U8W 进行仿真，首先必须将 U8W 设置为直通模式。U8W/U8W-Mini 实现 USB 转串口直通模式的方法如下：

1、 首先 U8W/U8W-Mini 固件必须升级到 v1.37 及以上版本

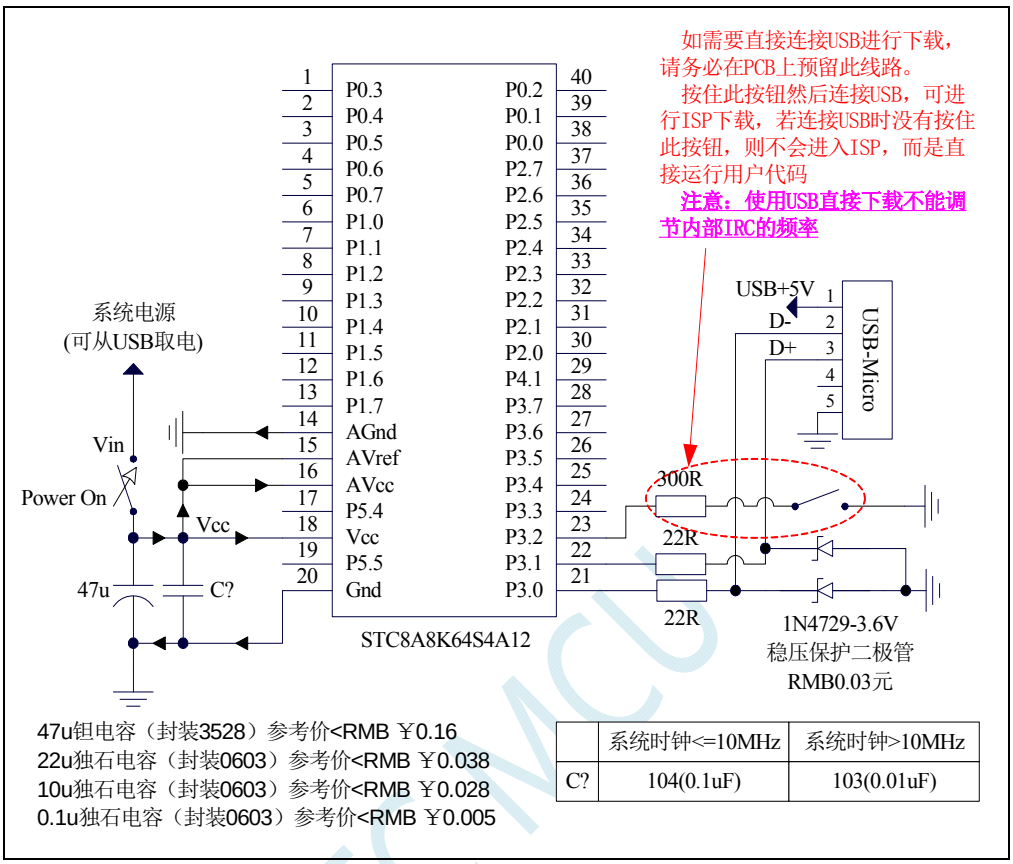
2、 U8W/U8W-Mini 上电后为正常下载模式，此时按住工具上的 Key1（下载）按键不要松开，再按一下 Key2（电源）按键，然后放开 Key2（电源）按键 后，再松开 Key1（下载）按键，U8W/U8W-Mini 会进入 USB 转串口直通模式。（**按下 Key1 ➔ 按下 Key2 ➔ 松开 Key2 ➔ 松开 Key1**）

3、 进入直通模式的 U8W/U8W-Mini 工具只是简单的 USB 转串口不具备脱机下载功能，若需要恢复 U8W/U8W-Mini 的原有功能，只需要再次单独按一下 Key2（电源）按键 即可

STC MCU

5.1.7 模拟USB直接ISP下载，仅支持ISP下载，不支持仿真

注：使用 USB 下载时需要将 P3.2 接 Gnd 才可进行正常下载



	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

ISP 下载步骤：

1、 将目标芯片停电

2、 P3.0/P3.1 按照如图所示的连接方式与 USB 端口连接好

3、 将 P3.2 与 GND 短接

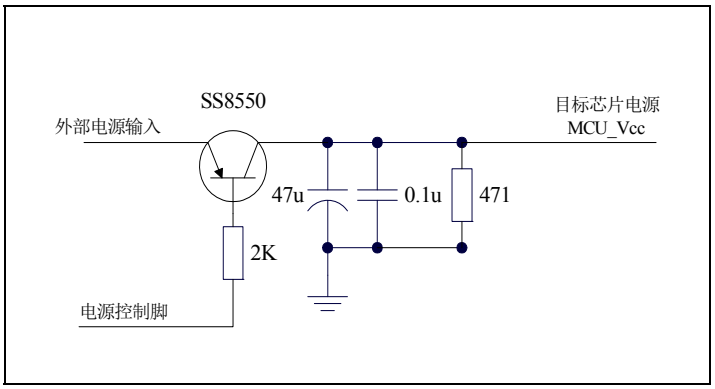
4、 给目标芯片上电，并等待 STC-ISP 下载软件中自动识别出“**STC USB Writer (USB1)**”

5、 点击下载软件中的“下载/编程”按钮（注意：与串口下载的操作顺序不同）

6、 开始 ISP 下载

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

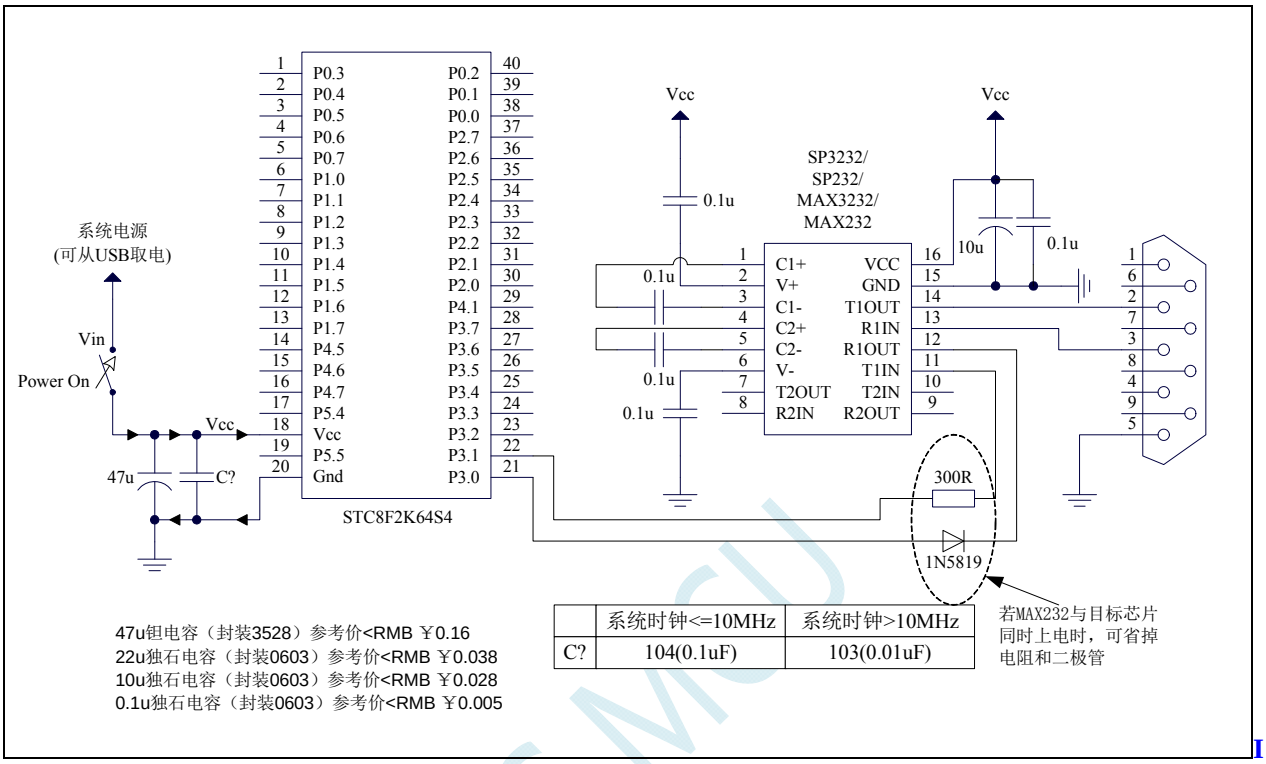
5.1.8 单片机电源控制参考电路



STC MCU

5.2 STC8F系列ISP下载应用线路图

5.2.1 使用RS-232 转换器下载，也可支持仿真



	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

SP 下载步骤：

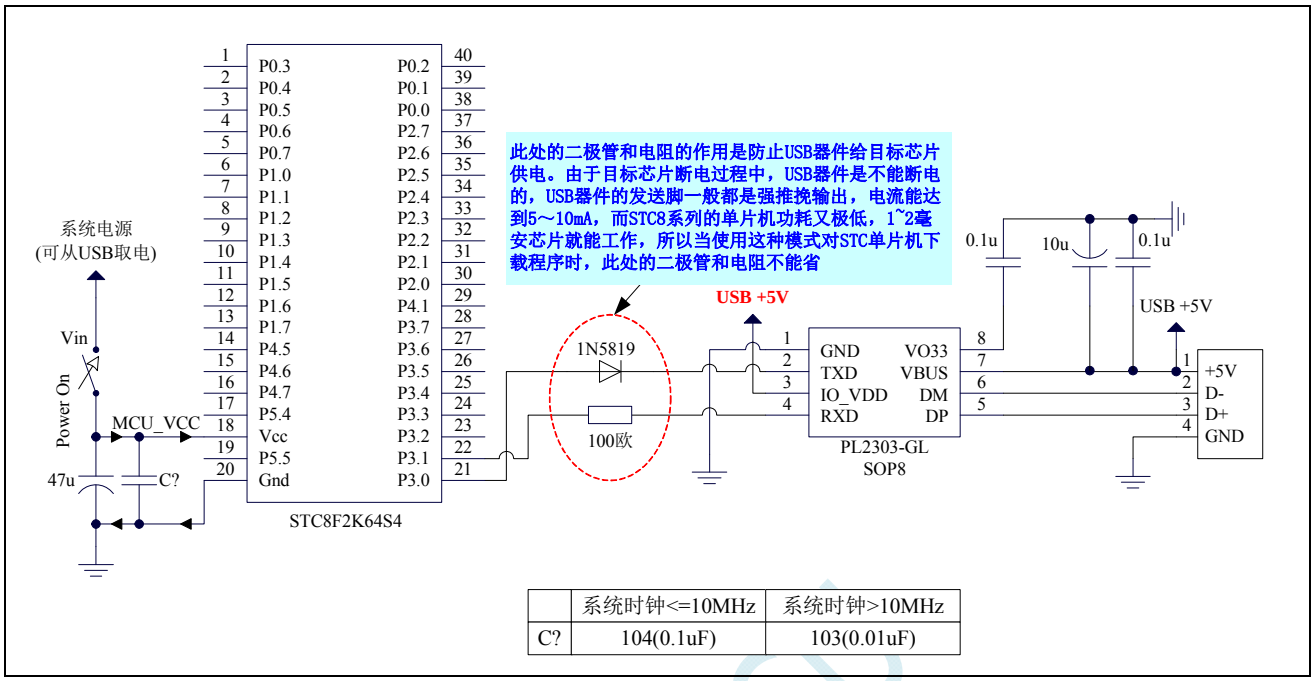
9、 给目标芯片停电

10、 点击 STC-ISP 下载软件中的“下载/编程”按钮

11、 给目标芯片上电

12、 开始 ISP 下载

5.2.2 使用PL2303-GL下载，也可支持仿真



	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

ISP 下载步骤：

6、 给目标芯片停电，注意不能给 USB 转串口芯片停电（如：CH340、PL2303-GL 等）

7、 由于 USB 转串口芯片的发送脚一般都是强推挽输出，必须在目标芯片的 P3.0 口和 USB 转串口芯片的发送脚之间串接一个二极管，否则目标芯片无法完全断电，达不到给目标芯片停电的目标。

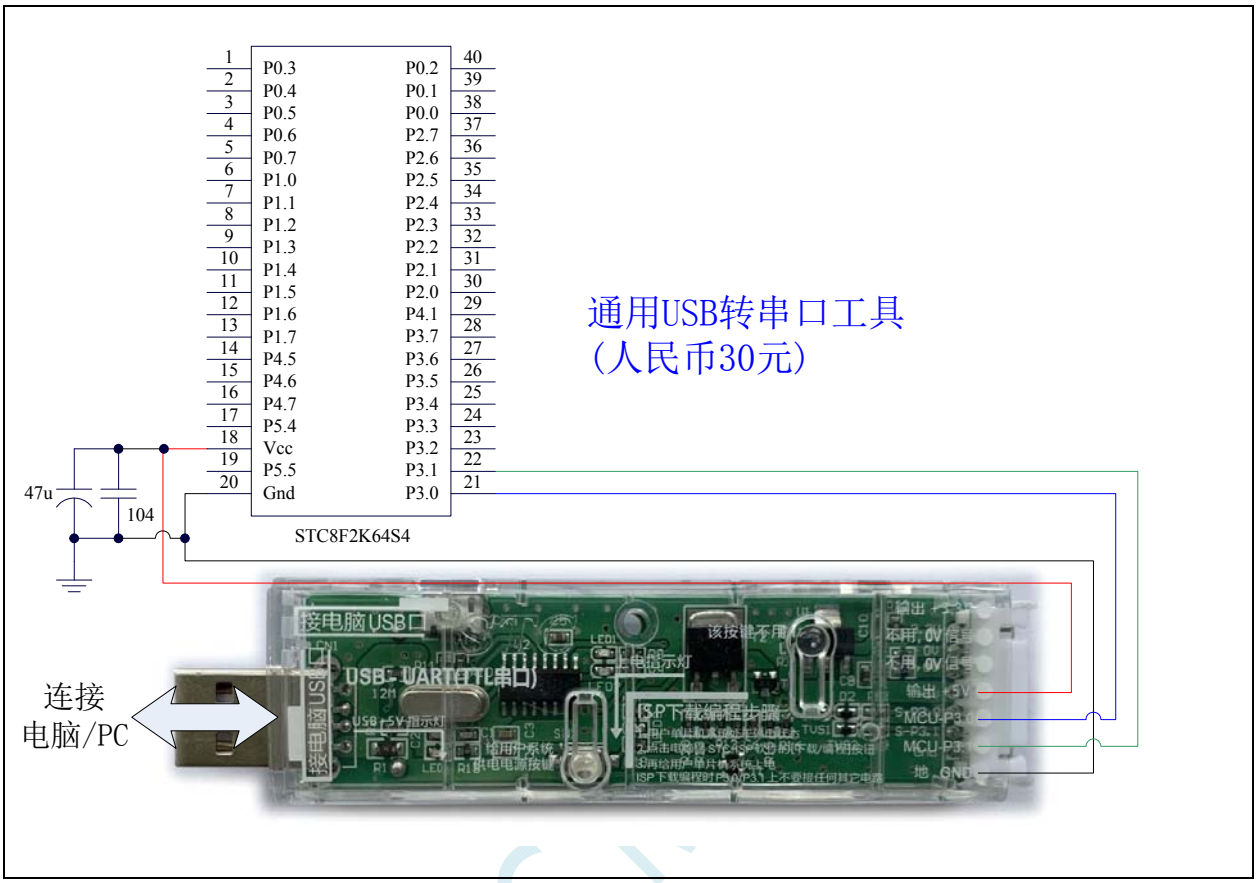
8、 点击 STC-ISP 下载软件中的“下载/编程”按钮

9、 给目标芯片上电

10、 开始 ISP 下载

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

5.2.3 使用通用USB转串口工具下载，支持ISP在线下载，也可支持仿真



ISP 下载步骤：

6、 按照如图所示的连接方式将通用 USB 转串口工具和目标芯片连接

7、 按下电源按钮，确定目标芯片处于**停电状态**（上电指示灯为灭的状态）。**注意：工具第一次上电时是不对外供电的，因此若是第一次上电使用此工具，可跳过此步。**

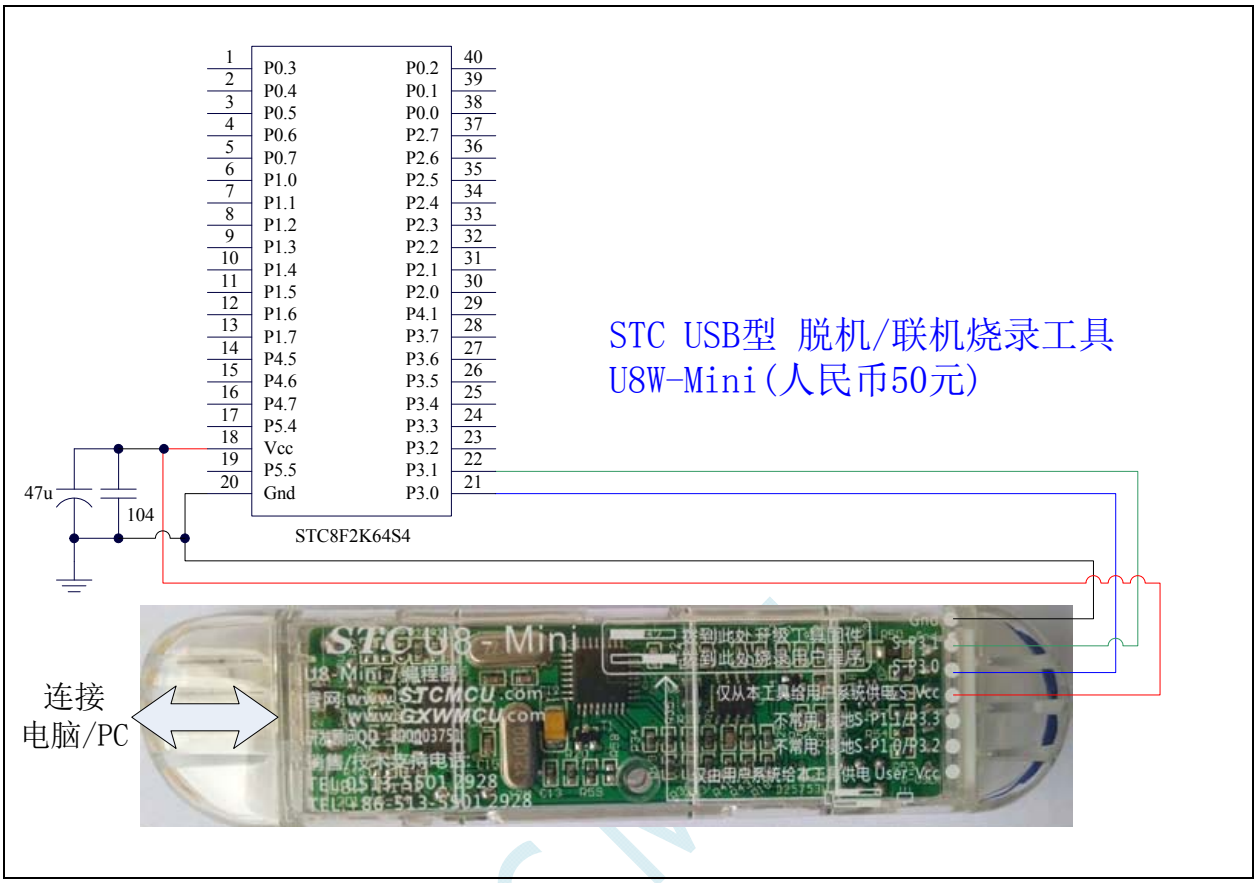
8、 点击 STC-ISP 下载软件中的“下载/编程”按钮

9、 再次按下电源按钮，给目标芯片上电（上电指示灯为亮的状态）

10、 开始 ISP 下载

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

5.2.4 使用U8-Mini工具下载，支持ISP在线和脱机下载，也可支持仿真



ISP 下载步骤：

4、 按照如图所示的连接方式将 U8-Mini 和目标芯片连接

5、 点击 STC-ISP 下载软件中的“下载/编程”按钮

6、 开始 ISP 下载

注意：若是使用 U8-Mini 给目标系统供电，目标系统的总电流不能大于 200mA，否则会导致下载失败。

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

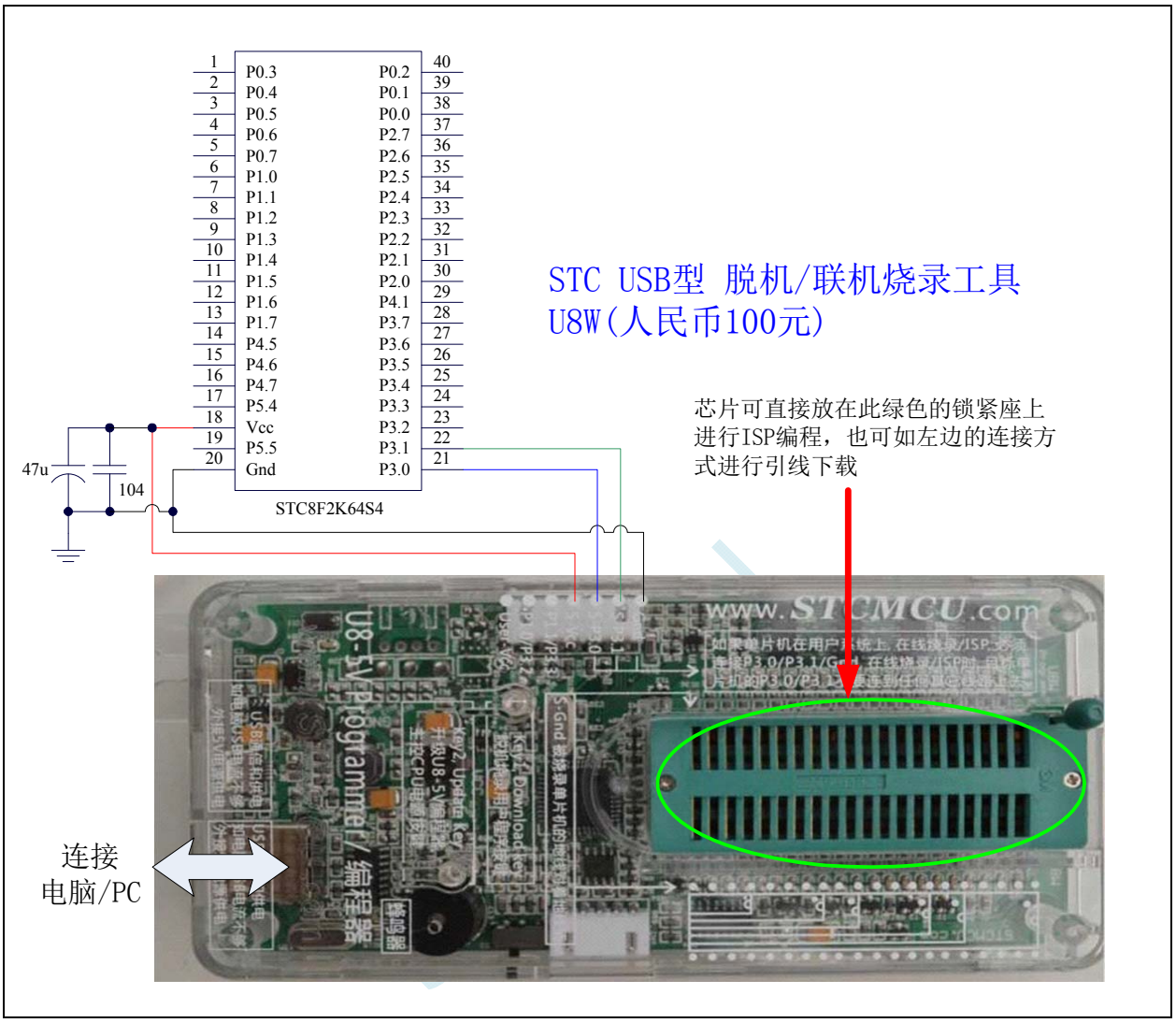
若要使用 U8-Mini 进行仿真，首先必须将 U8-Mini 设置为直通模式。U8W/U8W-Mini 实现 USB 转串口直通模式的方法如下：

4、 首先 U8W/U8W-Mini 固件必须升级到 v1.37 及以上版本

5、 U8W/U8W-Mini 上电后为正常下载模式，此时按住工具上的 Key1（下载）按键不要松开，再按一下 Key2（电源）按键，然后放开 Key2（电源）按键 后，再松开 Key1（下载）按键，U8W/U8W-Mini 会进入 USB 转串口直通模式。（**按下 Key1 ➔ 按下 Key2 ➔ 松开 Key2 ➔ 松开 Key1**）

6、 进入直通模式的 U8W/U8W-Mini 工具只是简单的 USB 转串口不具备脱机下载功能，若需要恢复 U8W/U8W-Mini 的原有功能，只需要再次单独按一下 Key2（电源）按键 即可

5.2.5 使用U8W工具下载，支持ISP在线和脱机下载，也可支持仿真



ISP 下载步骤（连线方式）：

4、 按照如图所示的连接方式将 U8W 和目标芯片连接

5、 点击 STC-ISP 下载软件中的“下载/编程”按钮

6、 开始 ISP 下载

注意：若是使用 U8W 给目标系统供电，目标系统的总电流不能大于 200mA，否则会导致下载失败。

ISP 下载步骤（在板方式）：

4、 将芯片按照 1 脚靠近锁紧扳手、管脚向下靠齐的方向放置好目标芯片

5、 点击 STC-ISP 下载软件中的“下载/编程”按钮

6、 开始 ISP 下载

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

若要使用 U8W 进行仿真，首先必须将 U8W 设置为直通模式。U8W/U8W-Mini 实现 USB 转串口直通模式的方法如下：

4、 首先 U8W/U8W-Mini 固件必须升级到 v1.37 及以上版本

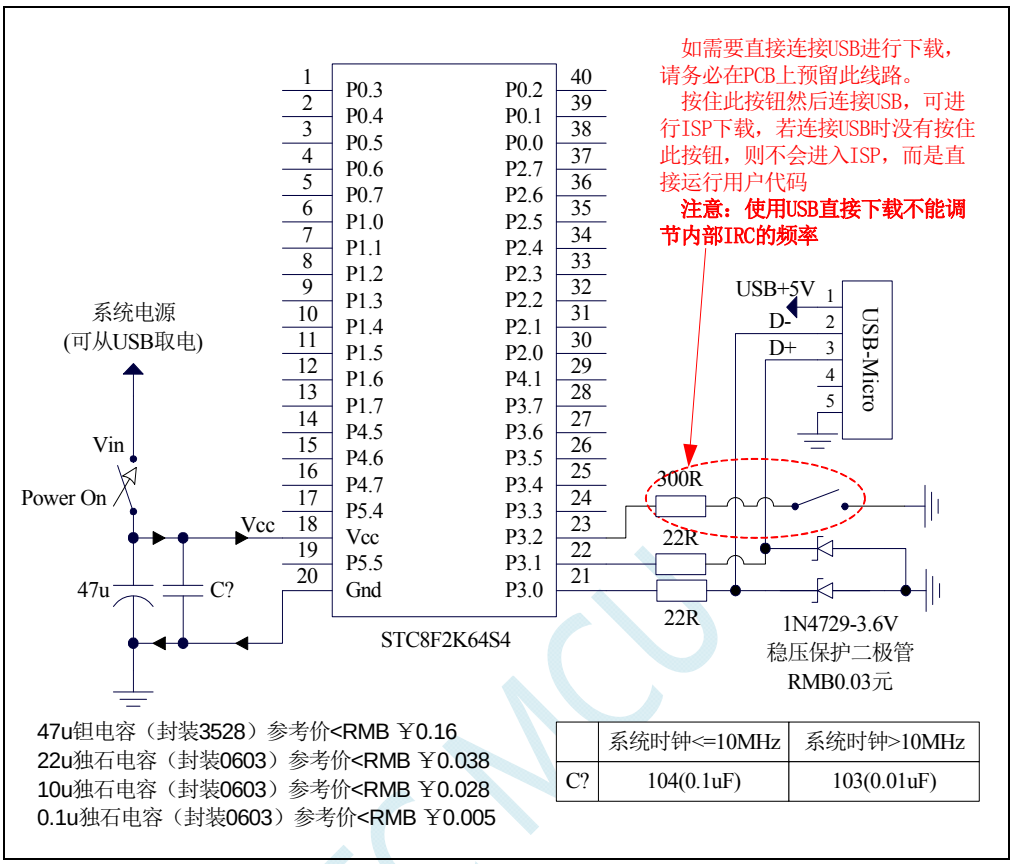
5、 U8W/U8W-Mini 上电后为正常下载模式，此时按住工具上的 Key1（下载）按键不要松开，再按一下 Key2（电源）按键，然后放开 Key2（电源）按键 后, 再松开 Key1（下载）按键，U8W/U8W-Mini 会进入 USB 转串口直通模式。（**按下 Key1 ➔ 按下 Key2 ➔ 松开 Key2 ➔ 松开 Key1**）

6、 进入直通模式的 U8W/U8W-Mini 工具只是简单的 USB 转串口不具备脱机下载功能，若需要恢复 U8W/U8W-Mini 的原有功能，只需要再次单独按一下 Key2（电源）按键 即可

STC MCU

5.2.6 模拟USB直接ISP下载，仅支持ISP下载，不支持仿真

注：使用 USB 下载时需要将 P3.2 接 Gnd 才可进行正常下载



	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

ISP 下载步骤：

7、 将目标芯片停电

8、 P3.0/P3.1 按照如图所示的连接方式与 USB 端口连接好

9、 将 P3.2 与 GND 短接

10、 给目标芯片上电，并等待 STC-ISP 下载软件中自动识别出“**STC USB Writer (USB1)**”

11、 点击下载软件中的“下载/编程”按钮（注意：与串口下载的操作顺序不同）

12、 开始 ISP 下载

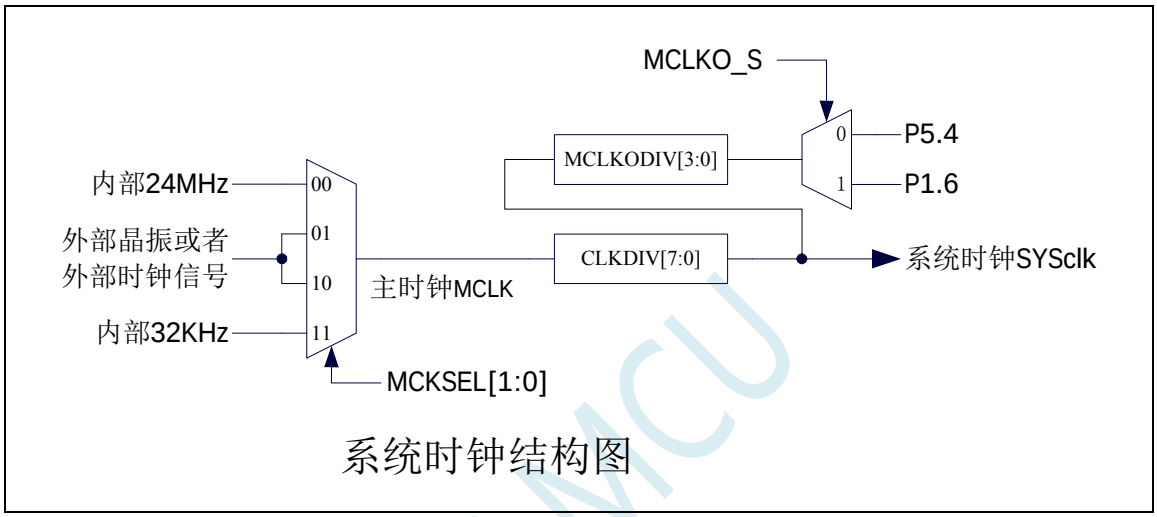
注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

6 时钟、复位、省电模式与系统电源管理

6.1 系统时钟控制

系统时钟控制器为单片机的 CPU 和所有外设系统提供时钟源，系统时钟有 3 个时钟源可供选择：内部高精度 24MHz 的 IRC、内部 32KHz 的 IRC（误差较大）、外部晶体振荡器或外部时钟信号。用户可通过程序分别使能和关闭各个时钟源，以及内部提供时钟分频以达到降低功耗的目的。

单片机进入掉电模式后，时钟控制器将会关闭所有的时钟源



系统时钟结构图

相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CKSEL	时钟选择寄存器	FE00H	MCLKODIV[3:0]				MCLKO_S	-	MCKSEL[1:0]		0000,0000
CLKDIV	时钟分频寄存器	FE01H									0000,0100
IRC24MCR	内部 24M 振荡器控制寄存器	FE02H	ENIRC24M	-	-	-	-	-	-	IRC24MST	1xxx,xxx0
XOSCCR	外部晶振控制寄存器	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST	00xx,xxx0
IRC32KCR	内部 32K 振荡器控制寄存器	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0

6.1.1 系统时钟选择寄存器（CKSEL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CKSEL	FE00H	MCLKODIV[3:0]				MCLKO_S	MCKSEL[1:0]		

MCLKODIV[3:0]：系统时钟输出分频系数

（**注意：系统时钟分频输出的时钟源是主时钟 MCLK 经过 CLKDIV 分频后的系统时钟**）

MCLKODIV[3:0]	系统时钟分频输出频率
0000	不输出时钟
0001	SYSClk/1
001x	SYSClk /2
010x	SYSClk /4

011x	SYSClk /8
100x	SYSClk /16
101x	SYSClk /32
110x	SYSClk /64
111x	SYSClk /128

MCLKO_S：系统时钟输出管脚选择

0：系统时钟分频输出到 P5.4 口

1：系统时钟分频输出到 P1.6 口

MCKSEL[1:0]：主时钟源选择

MCKSEL[1:0]	主时钟源
00	内部 24MHz 高精度 IRC
01	外部晶体振荡器或 外部输入时钟信号
10	
11	内部 32KHz 低速 IRC

6.1.2 时钟分频寄存器（CLKDIV）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	FE01H								

CLKDIV：主时钟分频系数。系统时钟 SYSCLK 是对主时钟 MCLK 进行分频后的时钟信号。

CLKDIV	系统时钟频率
0	MCLK/1
1	MCLK/1
2	MCLK/2
3	MCLK/3
…	…
x	MCLK/x
…	…
255	MCLK/255

6.1.3 内部 24M高精度IRC控制寄存器（IRC24MCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRC24MCR	FE02H	ENIRC24M	-	-	-	-	-	-	IRC24MST

ENIRC24M：内部 24M 高精度 IRC 使能位

0：关闭内部 24M 高精度 IRC

1：使能内部 24M 高精度 IRC

IRC24MST：内部 24M 高精度 IRC 频率稳定标志位。（只读位）

当内部 24M 的 IRC 从停振状态开始使能后，必须经过一段时间，振荡器的频率才会稳定，当振荡器频率稳定后，时钟控制器会自动将 IRC24MST 标志位置 1。所以当用户程序需要将时钟切换到使用内部 24M 的 IRC 时，首先必须设置 ENIRC24M=1 使能振荡器，然后一直查询振荡器稳定标志位 IRC24MST，直到标志位变为 1 时，才可进行时钟源切换。

6.1.4 外部振荡器控制寄存器（XOSCCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0

XOSCCR	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST

ENXOSC：外部晶体振荡器使能位

0：关闭外部晶体振荡器

1：使能外部晶体振荡器

XITYPE：外部时钟源类型

0：外部时钟源是外部时钟信号（或有源晶振）。信号源只需连接单片机的 XTALI（P1.7）

1：外部时钟源是晶体振荡器。信号源连接单片机的 XTALI（P1.7）和 XTALO（P1.6）

XOSCST：外部晶体振荡器频率稳定标志位。（只读位）

当外部晶体振荡器从停振状态开始使能后，必须经过一段时间，振荡器的频率才会稳定，当振荡器频率稳定后，时钟控制器会自动将 XOSCST 标志位置 1。所以当用户程序需要将时钟切换到使用外部晶体振荡器时，首先必须设置 ENXOSC=1 使能振荡器，然后一直查询振荡器稳定标志位 XOSCST，直到标志位变为 1 时，才可进行时钟源切换。

6.1.5 内部 32KHz低速IRC控制寄存器（IRC32KCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRC32KCR	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST

ENIRC32K：内部 32K 低速 IRC 使能位

0：关闭内部 32K 低速 IRC

1：使能内部 32K 低速 IRC

IRC32KST：内部 32K 低速 IRC 频率稳定标志位。（只读位）

当内部 32K 低速 IRC 从停振状态开始使能后，必须经过一段时间，振荡器的频率才会稳定，当振荡器频率稳定后，时钟控制器会自动将 IRC32KST 标志位置 1。所以当用户程序需要将时钟切换到使用内部 32K 低速 IRC 时，首先必须设置 ENIRC32K=1 使能振荡器，然后一直查询振荡器稳定标志位 IRC32KST，直到标志位变为 1 时，才可进行时钟源切换。

STCMCU

6.2 STC8A/8F系列内部IRC频率调整

STC8A/8F 系列单片机内部均集成有一颗高精度内部 IRC 振荡器。在用户使用 ISP 下载软件进行下载时，ISP 下载软件会根据用户所选择/设置的频率自动进行调整，一般频率值可调整到±0.3%以下，调整后的频率在全温度范围内（－40℃～85℃）的温漂可达-1.8%～0.8%。

STC8A/8F 系列内部 IRC 只有一个频段，此频段的中心频率约为 24MHz，最小频率约为 16MHz，最大频率约为 30MHz（注意：不同的芯片以及不同的生成批次可能会有约 5%左右的制造误差）。

内部 IRC 频率调整主要使用下面的 3 个寄存器进行调整

相关寄存器

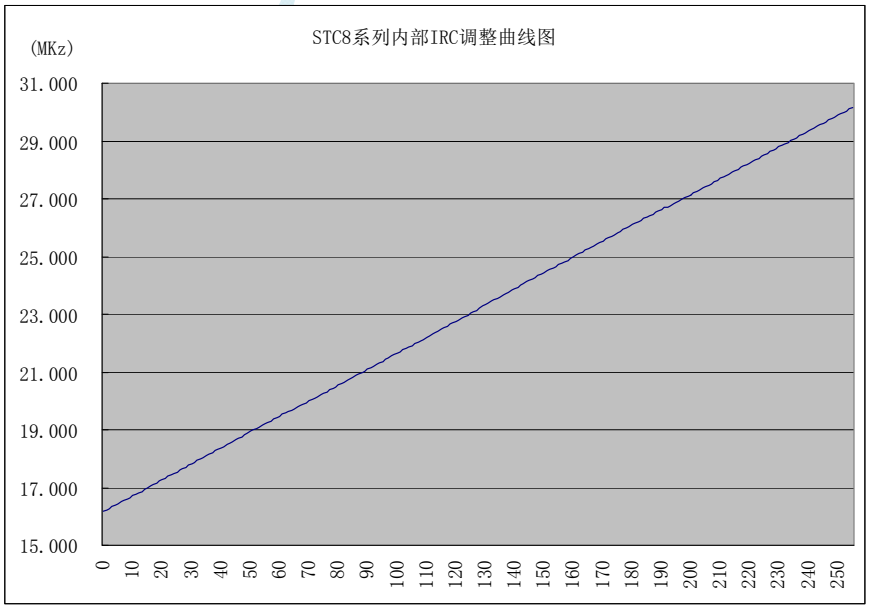
符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
LIRTRIM	IRC 频率微调寄存器	9EH	-	-	-	-	-	-	LIRTRIM[1:0]		0000,00nn
IRTRIM	IRC 频率调整寄存器	9FH	IRTRIM[7:0]								nnnn,nnnn
CLKDIV	时钟分频寄存器	FE01H	CLKDIV[7:0]								0000,0100

6.2.1 内部IRC频率调整寄存器（IRTRIM）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRTRIM	9FH	IRTRIM[7:0]							

IRTRIM[7:0]：内部高精度 IRC 频率调整寄存器

IRTRIM 可对 IRC 频率进行 256 个等级的调整，每个等级所调整的频率值在整体上呈线性分布，局部会有波动。宏观上，每一级所调整的频率约为 0.24%，即 IRTRIM 为（n+1）时的频率比 IRTRIM 为（n）时的频率约快 0.24%。但由于 IRC 频率调整并非每一级都是 0.24%（每一级所调整频率的最大值约为 0.55%，最小值约为 0.02%，整体平均值约为 0.24%），所以会造成局部波动。STC8A/8F 系列内部 IRC 频率调整的曲线图如下图所示：



6.2.2 内部IRC频率微调寄存器（LIRTRIM）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LIRTRIM	9EH	-	-	-	-	-	-	IRTRIM[1:0]	

LIRTRIM[1:0]：内部高精度 IRC 频率微调寄存器

LIRTRIM 可对 IRC 频率进行 3 个等级的调整，3 个等级所调整的频率范围如下表所示：

LIRTRIM[1:0]	调整的频率范围
00	不微调
01	调整约 0.10%
10	调整约 0.04%
11	调整约 0.10%

6.2.3 时钟分频寄存器（CLKDIV）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	FE01H								

CLKDIV：主时钟分频系数。系统时钟 SYSCLK 是对主时钟 MCLK 进行分频后的时钟信号。

CLKDIV	系统时钟频率
0	MCLK/1
1	MCLK/1
2	MCLK/2
3	MCLK/3
…	…
x	MCLK/x
…	…
255	MCLK/255

由于 STC8A/8F 系列内部只有一个以 24MHz 为中心频率的频段，频率范围只能在 16～30MHz，所以 STC8A/8F 系列内部 IRC 所能调整频率的上限即为 30MHz（由于制造误差，部分芯片的上限值可能为 28MHz～29MHz）；在未分频的情况下所能调整频率的下限为 16MHz（由于制造误差，部分芯片的下限值可能为 17MHz）。当用户需要使用低于 16MHz 的频率时，可将内部 IRC 的频率调整到目标频率的 2 倍或者 3 倍等，再使用 CLKDIV 进行分频即可得到用户所需要的频率。例如用户需要 11.0592MHz 的频率，使用内部 IRC 直接调整是无法得到这个频率的，但可将内部 IRC 调整到 22.1184MHz，在使用 CLKDIV 进行 2 分频即可得到 11.0592MHz。

注意：由于 STC8A/8F 系列内部 IRC 频段的最小频率比最大频率的 1/2 大，所以 STC8A/8F 系列在进行频率调整时存在盲区。一般 STC8A/8F 系列的芯片无法调整到 15MHz～16MHz 之间频率（由于制造误差，部分芯片的盲区可能会更大一些）。

6.3 STC15 系列内部IRC频率调整

STC15 系列单片机内部均集成有一颗高精度内部 IRC 振荡器。在用户使用 ISP 下载软件进行下载时，ISP 下载软件会根据用户所选择/设置的频率自动进行调整，一般频率值可调整到±0.3％以下，调整后的频率在全温度范围内（－40℃～85℃）的温漂可达±1％

STC15 系列内部 IRC 有 4 个频段，4 个频段的中心频率分别约为 5.5MHz、11MHz、22MHz 和 33MHz，其中 5.5MHz 频段的频率范围约在 3.8MHz～7.1MHz，11MHz 频段的频率范围约在 7.5MHz～13.8MHz，22MHz 频段的频率范围约在 15.5MHz～28MHz，33MHz 频段的频率范围约在 24.3MHz～43MHz（注意：不同的芯片以及不同的生成批次可能会有约 5％左右的制造误差）。

注意：STC15 系列内部 IRC 的 4 个频段的选择必须通过 ISP 下载时进行自动选择，在用户程序中只能使用特殊功能寄存器 IRTRIM 在同一频段内进行微调。

STC15 系列内部 IRC 频率调整主要使用下面的寄存器进行调整

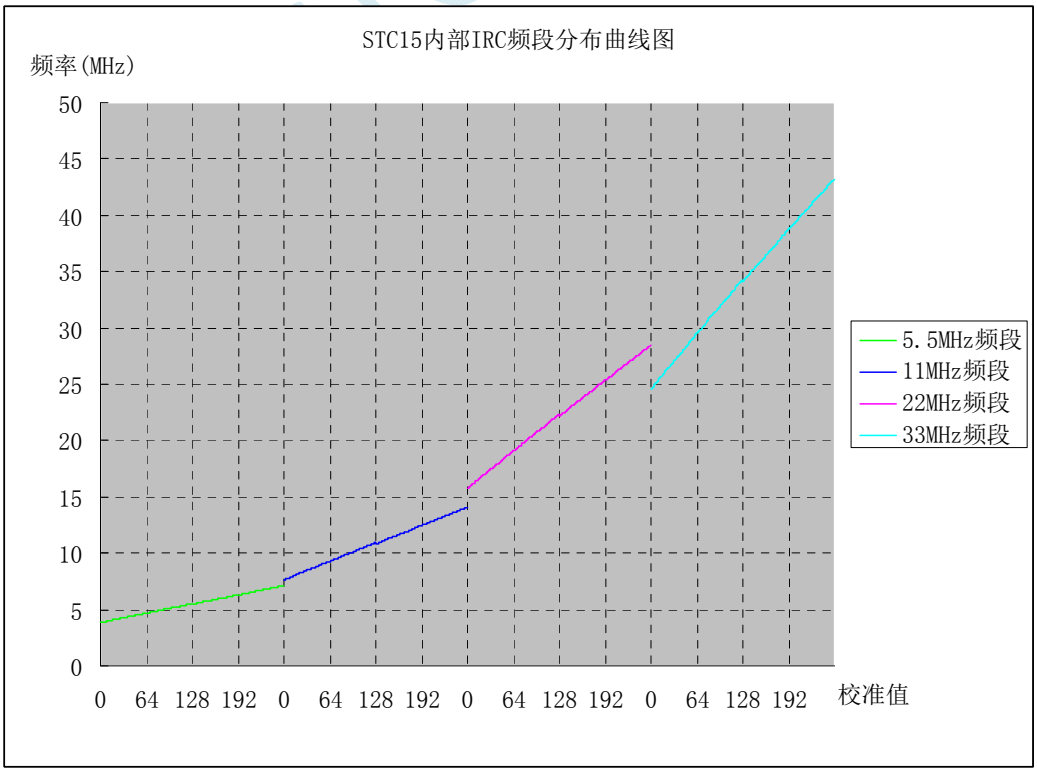
内部 IRC 频率调整寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRTRIM	9FH	IRTRIM[7:0]							

IRTRIM[7:0]：内部高精度 IRC 频率调整寄存器

IRTRIM 可对 IRC 每个频段的频率进行 256 个等级的调整（在进行频率调整前必须先选择相应的目标频段），每个等级所调整的频率值在整体上呈线性分布，局部会有波动。宏观上，每一级所调整的频率约为 0.24％，即 IRTRIM 为（n+1）时的频率比 IRTRIM 为（n）时的频率约快 0.24％。但由于 IRC 频率调整并非每一级都是 0.24％（每一级所调整频率的最大值约为 0.55％，最小值约为 0.02％，整体平均值约为 0.24％），所以会造成局部波动。

STC15 系列内部 IRC 各个频段的频率调整的曲线图如下图所示：



从上图可用看出，STC15 系列的 4 个频段中，存在频率盲区。

5.5MHz 和 11MHz 两个频段的过渡点中存在断层，即 5.5MHz 频段的可调最大频率值比 11MHz 频段的可调最小频率值小，所以 **STC15 系列在进行频率调整时 5.5MHz 和 11MHz 之间存在频率盲区（一般在 7MHz~8MHz 之间，由于制造误差，部分芯片的盲区可能会更大一些）**。

同样，11MHz 和 22MHz 两个频段的过渡点中存在断层，即 11MHz 频段的可调最大频率值比 22MHz 频段的可调最小频率值小，所以 **STC15 系列在进行频率调整时 11MHz 和 22MHz 之间存在频率盲区（一般在 13MHz~16MHz 之间，由于制造误差，部分芯片的盲区可能会更大一些）**；

22MHz 与 33MHz 两个频段的过渡点中不存在断层，故也不存在频率盲区。

由于有盲区的存在，所以用户在给目标芯片设置 IRC 频率时，应注意尽量避免使用可能出现在盲区的频率。

STC MCU

6.4 系统复位

STC8A/8F 系列单片机的复位分为硬件复位和软件复位两种。

硬件复位时，所有的寄存器的值会复位到初始值，系统会重新读取所有的硬件选项。同时根据硬件选项所设置的上电等待时间进行上电等待。硬件复位主要包括：

- 上电复位
- 低压复位
- 复位脚复位
- 看门狗复位

软件复位时，除与时钟相关的寄存器保持不变外，其余的所有寄存器的值会复位到初始值，软件复位不会重新读取所有的硬件选项。软件复位主要包括：

- 写 IAP_CONTR 的 SWRST 所触发的复位

相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
WDT_CONTR	看门狗控制寄存器	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]			0x00,0000
IAP_CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	IAP_WT[2:0]			0000,x000
RSTCFG	复位配置寄存器	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]		0000,0000

STC MCU

6.4.1 看门狗复位（WDT_CONTR）

在工业控制/汽车电子/航空航天等需要高可靠性的系统中，为了防止“系统在异常情况下，受到干扰，MCU/CPU 程序跑飞，导致系统长时间异常工作”，通常是引进看门狗，如果 MCU/CPU 不在规定的时间内按要求访问看门狗，就认为 MCU/CPU 处于异常状态，看门狗就会强制 MCU/CPU 复位，使系统重新从头开始执行用户程序。

STC8 系列的看门狗复位是热启动复位中的硬件复位之一。STC8 系列单片机引进此功能，使单片机系统可靠性设计变得更加方便、简洁。STC8 系列看门狗复位状态结束后，系统固定从 ISP 监控程序区启动，与看门狗复位前 IAP_CONTR 寄存器的 SWBS 无关（**注意：此处与 STC15 系列 MCU 不同**）

WDT_CONTR（看门狗控制寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
WDT_CONTR	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]		

WDT_FLAG：看门狗溢出标志

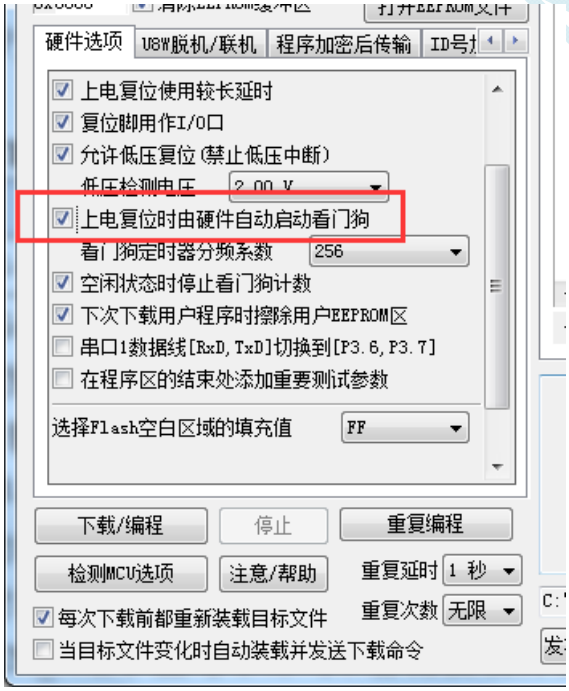
看门狗发生溢出时，硬件自动将此位置 1，需要软件清零。

EN_WDT：看门狗使能位

0：对单片机无影响

1：启动看门狗定时器。

注意：看门狗定时器可使用软件方式启动，也可硬件自动启动，一旦看门狗定时器启动后，软件将无法关闭，必须对单片机进行重新上电才可关闭。软件启动看门狗只需要对 EN_WDT 位写 1 即可。若需要硬件启动看门狗，则需要在 ISP 下载时进行如下图所示的设置：



CLR_WDT：看门狗定时器清零

0：对单片机无影响

1：清零看门狗定时器，硬件自动将此位复位

IDL_WDT：IDLE 模式时的看门狗控制位

0：IDLE 模式时看门狗停止计数

1：IDLE 模式时看门狗继续计数

WDT_PS[2:0]：看门狗定时器时钟分频系数

WDT_PS[2:0]	分频系数	12M 主频时的溢出时间	20M 主频时的溢出时间
000	2	≈ 65.5 毫秒	≈ 39.3 毫秒
001	4	≈ 131 毫秒	≈ 78.6 毫秒
010	8	≈ 262 毫秒	≈ 157 毫秒
011	16	≈ 524 毫秒	≈ 315 毫秒
100	32	≈ 1.05 秒	≈ 629 毫秒
101	64	≈ 2.10 秒	≈ 1.26 秒
110	128	≈ 4.20 秒	≈ 2.52 秒
111	256	≈ 8.39 秒	≈ 5.03 秒

看门狗溢出时间计算公式如下：

$$\text{看门狗溢出时间} = \frac{12 \times 32768 \times 2^{(\text{WDT_PS}+1)}}{\text{SYSclk}}$$

STC MCU

6.4.2 软件复位（IAP_CONTR）

IAP_CONTR（IAP 控制寄存器） **对 IAP 控制寄存器写 60H，可达到对单片机冷启动的效果**

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	IAP_CMD[2:0]		

SWBS：软件复位启动选择

0：软件复位后从用户程序区开始执行代码。用户数据区的数据保持不变。

1：软件复位后从系统 ISP 区开始执行代码。用户数据区的数据会被初始化。

SWRST：软件复位触发位

0：对单片机无影响

1：触发软件复位

STC MCU

6.4.3 低压复位（RSTCFG）

RSTCFG（复位配置寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
RSTCFG	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]	

ENLVR：低压复位控制位

0：禁止低压复位。当系统检测到低压事件时，会产生低压中断

1：使能低压复位。当系统检测到低压事件时，自动复位

P54RST：RST 管脚功能选择

0：RST 管脚用作普通 I/O 口（P5.4）

1：RST 管脚用作复位脚

LVDS[1:0]：低压检测门槛电压设置

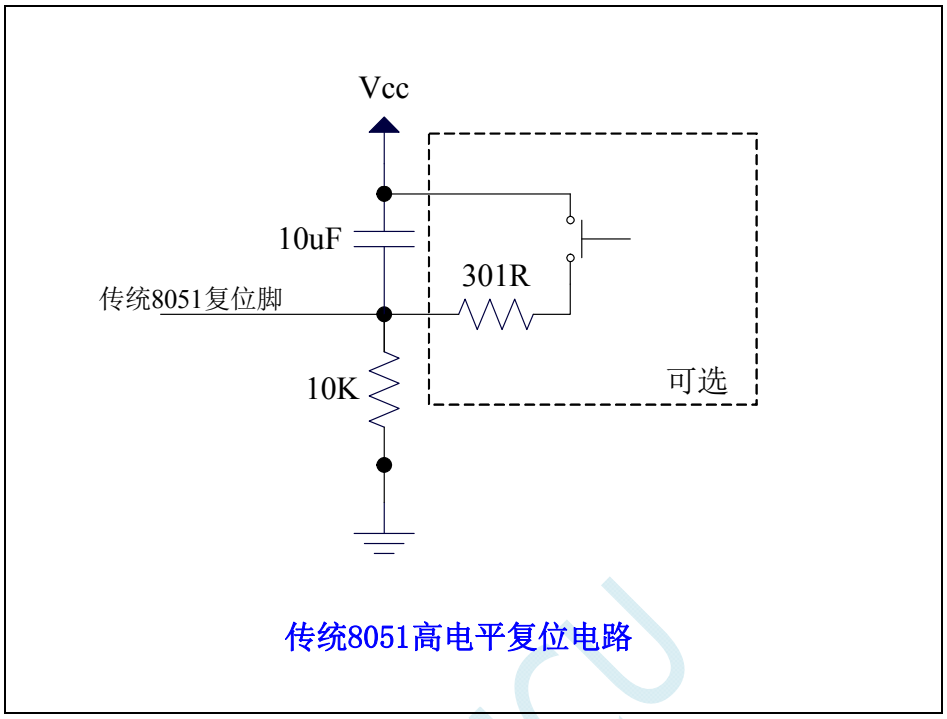
LVDS[1:0]	低压检测门槛电压
00	2.2V
01	2.4V
10	2.7V
11	3.0V

注意事项：

1. **复位脚用于复位时，下拉电阻不大于 3K。在 1.5V 时有超过 200uA 的上拉电流，10K 电路拉不低。**
2. **使用外部晶振时，要注意下面的要求： AVCC 跟 VCC 电压差不要超过 0.1V，P1.0~P1.7 P0.0~P0.6 电压不能比 AVCC 高，否则晶振不起振，形同死机。**

STCMCU

6.4.4 传统 8051 高电平上电复位参考电路



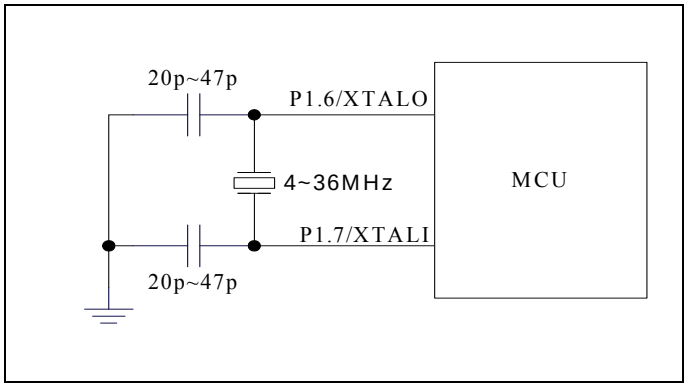
传统8051高电平复位电路

上图为传统 8051 的高电平复位电路，STC8A/8F 可参考此复位电路

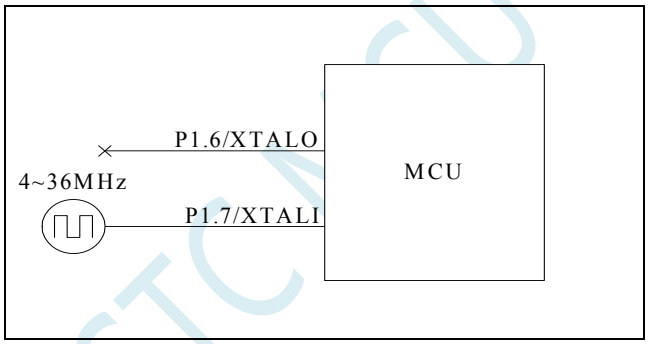
STC MCU

6.5 外部晶振及外部时钟电路

6.5.1 外部晶振输入电路



6.5.2 外部时钟输入电路（P1.6 不可用作普通I/O）



6.6 时钟停振/省电模式与系统电源管理

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
VOCTRL	电压控制寄存器	BBH	SCC	-	-	-	-	-	0	0	0xxx,xx00

6.6.1 电源控制寄存器（PCON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

LVDF：低压检测标志位。当系统检测到低压事件时，硬件自动将此位置 1，并向 CPU 提出中断请求。此位需要用户软件清零。

POF：上电复位标志位。MCU 每次重新上电后，硬件自动将此位置 1，可软件将此位清零。

PD：时钟停振模式/掉电模式/停电模式控制位

0：无影响

1：单片机进入时钟停振模式/掉电模式/停电模式，CPU 以及全部外设均停止工作。唤醒后硬件自动

清零。（**注：时钟停振模式下，CPU 和全部的外设均停止工作，但 SRAM 和 XRAM 中的数据是一直维持不变的**）

IDL：IDLE（空闲）模式控制位

0：无影响

1：单片机进入 IDLE 模式，只有 CPU 停止工作，其他外设依然在运行。唤醒后硬件自动清零

注：虽然 LVD 和比较器均可唤醒时钟停振模式化，但时钟停振省电模式下，不建议启动 LVD 和比较器，否则硬件系统还会自动启动内部 1.344V 的高精准参考源，这个高精准参考源有有相应的抗温漂和调校线路，大约会额外增加 300uA 的耗电，而 MCU 进入时钟停振模式后，3.3V 工作电压时只耗约 0.4uA 的电流，所以进入时钟停振模式时不建议开 LVD 和比较器。如果确实需要用，建议开启掉电唤醒定时器，掉电唤醒定时器只会增加约 1.4uA 的耗电，这个耗电一般系统是可以接受的。让掉电唤醒定时器每 5 秒唤醒一次 MCU，唤醒后可用 LVD、比较器、 ADC 检测外部电池电压，检测工作约耗时 1mS 后再进入时钟停振/省电模式，这样增加的平均电流小于 1uA，则整体功耗大约为 2.8uA（0.4uA + 1.4uA + 1uA）。

可将 CPU 从掉电模式唤醒的外部管脚有：INT0/P3.2、INT1/P3.3、INT2/P3.6、INT3/P3.7、INT4/P3.0、T0/P3.4、T1/P3.5、T2/P1.2、T3/P0.4、T4/P0.6、CCP0/P1.7、CCP1/P1.6、CCP2/P1.5、CCP4/P1.4、CCP0_2/P2.3、CCP1_2/P2.4、CCP2_2/P2.5、CCP3_2/P2.6、CCP0_3/P7.0、CCP1_3/P7.1、CCP2_3/P7.2、CCP3_3/P7.3、CCP0_4/P3.3、CCP1_4/P3.2、CCP2_4/P3.1、CCP3_4/P3.0、RxD/P3.0、RxD_2/P3.6、RxD_3/P1.6、RxD_4/P4.3、RxD2/P1.0、RxD2_2/P4.0、RxD3/P0.0、RxD3_2/P5.0、RxD4/P0.2、RxD4_2/P5.2，另外还有掉电唤醒定时器、低电压中断和比较器中断也可将 CPU 从掉电模式唤醒。

6.6.2 电压控制寄存器（VOCTRL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
VOCTRL	BBH	SCC						0	0

SCC：静态电流控制位

0：选择内部静态保持电流控制线路，静态电流一般为 1.5uA 左右。

1：选择外部静态保持电流控制线路，选择此模式时功耗更低。此模式下 STC8A8K 系列的静态电流一般为 0.15uA 以下；STC8F2K 系列的的静态电流一般为 0.1uA 以下。注意：选择此模式进入掉电模式后，VCC 管脚的电压不能有较大波动，否则对 MCU 内核可能会有不良影响。

[B1:B0]：内部测试位，必须写入 0

6.7 掉电唤醒定时器

内部掉电唤醒定时器是一个 15 位的计数器（由{WKTCH[6:0],WKTCL[7:0]}组成 15 位）。用于唤醒处于掉电模式的 MCU。

6.7.1 掉电唤醒定时器计数寄存器（WKTCL，WKTCH）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
WKTCL	AAH								
WKTCH	ABH	WKTEN							

WKTEN：掉电唤醒定时器的使能控制位

0：停用掉电唤醒定时器

1：启用掉电唤醒定时器

如果 STC8A/8F 系列单片机内置掉电唤醒专用定时器被允许（通过软件将 WKTCH 寄存器中的 WKTEN 位置 1），当 MCU 进入掉电模式/停机模式后，掉电唤醒专用定时器开始计数，当计数值与用户所设置的值相等时，掉电唤醒专用定时器将 MCU 唤醒。MCU 唤醒后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行。掉电唤醒之后，可以通过读 WKTCH 和 WKTCL 中的内容获取单片机在掉电模式中的睡眠时间。

这里请注意：用户在寄存器{WKTCH[6:0],WKTCL[7:0]}中写入的值必须比实际计数值少 1。如用户需计数 10 次，则将 9 写入寄存器{WKTCH[6:0],WKTCL[7:0]}中。同样，如果用户需计数 32767 次，则应对{WKTCH[6:0],WKTCL[7:0]}写入 7FFEH（即 32766）。（**计数值 0 和计数值 32767 为内部保留值，用户不能使用**）。内部掉电唤醒定时器有自己的内部时钟，掉电唤醒定时器计数一次的时间就是由该时钟决定的。内部掉电唤醒定时器的时钟频率约为 32KHz，误差较大。用户可以通过读 RAM 区 F8H 和 F9H 的内容（F8H 存放频率的高字节，F9H 存放低字节）来获取内部掉电唤醒专用定时器出厂时所记录的时钟频率。

掉电唤醒专用定时器计数时间的计算公式如下所示：（F_{wt} 为我们从 RAM 区 F8H 和 F9H 获取到的内部掉电唤醒专用定时器的时钟频率）

$$\text{掉电唤醒定时器定时时间} = \frac{10^6 \times 16 \times \text{计数次数}}{F_{wt}} \text{(微秒)}$$

假设 F_{wt}=32KHz，则有：

{WKTCH[6:0],WKTCL[7:0]}	掉电唤醒专用定时器计数时间
~~0（内部保留）~~	
1	$10^6 \div 32K \times 16 \times (1+1) \approx 1$ 毫秒
9	$10^6 \div 32K \times 16 \times (1+9) \approx 5$ 毫秒
99	$10^6 \div 32K \times 16 \times (1+99) \approx 50$ 毫秒
999	$10^6 \div 32K \times 16 \times (1+999) \approx 0.5$ 秒
4095	$10^6 \div 32K \times 16 \times (1+4095) \approx 2$ 秒
32766	$10^6 \div 32K \times 16 \times (1+32766) \approx 16$ 秒
~~32767（内部保留）~~	

6.8 范例程序

6.8.1 选择系统时钟源

汇编代码

```
P_SW2       DATA    0BAH
CKSEL       EQU     0FE00H
CLKDIV      EQU     0FE01H
IRC24MCR    EQU     0FE02H
XOSCCR      EQU     0FE03H
IRC32KCR    EQU     0FE04H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH

            MOV     P_SW2,#80H
            MOV     A,#00H              ;选择内部 IRC ( 默认 )
            MOV     DPTR,#CKSEL
            MOVX    @DPTR,A
            MOV     P_SW2,#00H

;           MOV     P_SW2,#80H
;           MOV     A,#0C0H             ;启动外部晶振
;           MOV     DPTR,#XOSCCR
;           MOVX    @DPTR,A
;           MOVX    A,@DPTR
;           JNB     ACC.0,$-1           ;等待时钟稳定
;           CLR     A                   ;时钟不分频
;           MOV     DPTR,#CLKDIV
            MOVX    @DPTR,A
;           MOV     A,#01H              ;选择外部晶振
;           MOV     DPTR,#CKSEL
;           MOVX    @DPTR,A
;           MOV     P_SW2,#00H

;           MOV     P_SW2,#80H
;           MOV     A,#80H              ;启动内部 32K IRC
;           MOV     DPTR,#IRC32KCR
;           MOVX    @DPTR,A
;           MOVX    A,@DPTR
;           JNB     ACC.0,$-1           ;等待时钟稳定
;           CLR     A                   ;时钟不分频
;           MOV     DPTR,#CLKDIV
;           MOVX    @DPTR,A
;           MOV     A,#03H              ;选择内部 32K
;           MOV     DPTR,#CKSEL
;           MOVX    @DPTR,A
;           MOV     P_SW2,#00H

            JMP     $

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

#define   CKSEL       (*(unsigned char volatile xdata *)0xfe00)
#define   CLKDIV      (*(unsigned char volatile xdata *)0xfe01)
#define   IRC24MCR    (*(unsigned char volatile xdata *)0xfe02)
#define   XOSCCR      (*(unsigned char volatile xdata *)0xfe03)
#define   IRC32KCR    (*(unsigned char volatile xdata *)0xfe04)

sfr       P_SW2       =   0xba;

void main()
{
    P_SW2 = 0x80;
    CKSEL = 0x00;                   //选择内部 IRC ( 默认 )
    P_SW2 = 0x00;

/*
    P_SW2 = 0x80;
    XOSCCR = 0xc0;                  //启动外部晶振
    while (!(XOSCCR & 1));          //等待时钟稳定
    CLKDIV = 0x00;                  //时钟不分频
    CKSEL = 0x01;                   //选择外部晶振
    P_SW2 = 0x00;
*/

/*
    P_SW2 = 0x80;
    IRC32KCR = 0x80;                //启动内部 32K IRC
    while (!(IRC32KCR & 1));        //等待时钟稳定
    CLKDIV = 0x00;                  //时钟不分频
    CKSEL = 0x03;                   //选择内部 32K
    P_SW2 = 0x00;
*/
    while (1);
}
```

6.8.2 主时钟分频输出

汇编代码

```
P_SW2       DATA    0BAH
CKSEL       EQU     0FE00H
CLKDIV      EQU     0FE01H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH

            MOV     P_SW2,#80H
;           MOV     A,#10H          ;主时钟输出到 P5.4 口
;           MOV     A,#20H          ;主时钟 2 分频输出到 P5.4 口
            MOV     A,#40H          ;主时钟 4 分频输出到 P5.4 口
```

```
;               MOV     A,#48H                  ;主时钟 4 分频输出到 P1.6 口
                MOV     DPTR,#CKSEL
                MOVX    @DPTR,A
                MOV     P_SW2,#00H

                JMP     $

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

#define     CKSEL           (*(unsigned char volatile xdata *)0xfe00)
#define     CLKDIV          (*(unsigned char volatile xdata *)0xfe01)

sfr         P_SW2       =   0xba;

void main()
{
    P_SW2 = 0x80;
//  CKSEL = 0x10;                   //主时钟输出到 P5.4 口
//  CKSEL = 0x20;                   //主时钟 2 分频输出到 P5.4 口
    CKSEL = 0x40;                   //主时钟 4 分频输出到 P5.4 口
//  CKSEL = 0x48;                   //主时钟 4 分频输出到 P1.6 口
    P_SW2 = 0x00;

    while (1);
}
```

6.8.3 看门狗定时器应用

汇编代码

```
;测试工作频率为 11.0592MHz

WDT_CONTR   DATA    0C1H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH

;           MOV     WDT_CONTR,#23H      ;使能看门狗,溢出时间约为 0.5s
            MOV     WDT_CONTR,#24H      ;使能看门狗,溢出时间约为 1s
;           MOV     WDT_CONTR,#27H      ;使能看门狗,溢出时间约为 8s
            CLR     P3.2                ;测试端口
LOOP:
;           ORL     WDT_CONTR,#10H      ;清看门狗,否则系统复位
            JMP     LOOP

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为11.0592MHz

sfr     WDT_CONTR   =   0xc1;
sbit    P32         =   P3^2;

void main()
{
//  WDT_CONTR = 0x23;           //使能看门狗,溢出时间约为0.5s
    WDT_CONTR = 0x24;           //使能看门狗,溢出时间约为1s
//  WDT_CONTR = 0x27;           //使能看门狗,溢出时间约为8s
    P32 = 0;                    //测试端口

    while (1)
    {
//      WDT_CONTR |= 0x10;      //清看门狗,否则系统复位
    }
}
```

6.8.4 软复位实现自定义下载

汇编代码

```
;测试工作频率为11.0592MHz

IAP_CONTR   DATA    0C7H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH

            SETB    P3.2
            SETB    P3.3
LOOP:
            JB      P3.2,LOOP
            JB      P3.3,LOOP
            MOV     IAP_CONTR,#60H      ;检查到P3.2和P3.3同时为0时复位到ISP
            JMP     $

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为11.0592MHz

sfr     IAP_CONTR   =   0xc7;
sbit    P32         =   P3^2;
sbit    P33         =   P3^3;

void main()
```

```
{
    P32 = 1;                            //测试端口
    P33 = 1;                            //测试端口

    while (1)
    {
        if (!P32 && !P33)
        {
            IAP_CONTR |= 0x60;          //检查到 P3.2 和 P3.3 同时为 0 时复位到 ISP
        }
    }
}
```

6.8.5 低压检测

汇编代码

```
RSTCFG      DATA    0FFH
ENLVR       EQU     40H                 ;RSTCFG.6
LVD2V2      EQU     00H                 ;LVD@2.2V
LVD2V4      EQU     01H                 ;LVD@2.4V
LVD2V7      EQU     02H                 ;LVD@2.7V
LVD3V0      EQU     03H                 ;LVD@3.0V
ELVD        BIT     IE.6
LVDF        EQU     20H                 ;PCON.5

            ORG     0000H
            LJMP    MAIN
            ORG     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
            ANL     PCON,#NOT LVDF      ;清中断标志
            CPL     P3.2                ;测试端口
            RETI
MAIN:
            MOV     SP,#3FH

            ANL     PCON,#NOT LVDF      ;上电后需要先清 LVDF 标志
;           MOV     RSTCFG,#ENLVR | LVD3V0  ;使能 3.0V 时低压复位,不产生 LVD 中断
            MOV     RSTCFG,#LVD3V0      ;使能 3.0V 时低压中断
            SETB    ELVD                ;使能 LVD 中断
            SETB    EA
            JMP     $

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

sfr     RSTCFG      =   0xff;
#define ENLVR       0x40    //RSTCFG.6
#define LVD2V2      0x00    //LVD@2.2V
#define LVD2V4      0x01    //LVD@2.4V
#define LVD2V7      0x02    //LVD@2.7V
```

```
#define   LVD3V0          0x03      //LVD@3.0V
sbit      ELVD      =     IE^6;
#define   LVDF            0x20      //PCON.5
sbit      P32       =     P3^2;

void Lvd_Isr() interrupt 6
{
    PCON &= ~LVDF;                  //清中断标志
    P32 = ~P32;                     //测试端口
}

void main()
{
    PCON &= ~LVDF;                  //测试端口
//  RSTCFG = ENLVR | LVD3V0;        //使能 3.0V 时低压复位,不产生 LVD 中断
    RSTCFG = LVD3V0;                //使能 3.0V 时低压中断
    ELVD = 1;                       //使能 LVD 中断
    EA = 1;

    while (1);
}
```

6.8.6 省电模式

汇编代码

```
VOCTRL      DATA    0BBH
IDL         EQU     01H                 ;PCON.0
PD          EQU     02H                 ;PCON.1

            ORG     0000H
            LJMP    MAIN
            ORG     0003H
            LJMP    INT0ISR

            ORG     0100H
INT0ISR:
            CPL     P3.4                ;测试端口
            RETI
MAIN:
            MOV     SP,#3FH

            MOV     VOCTRL,#00H         ;掉电模式时使用内部 SCC 模块,功耗约 1.5uA
;           MOV     VOCTRL,#80H         ;掉电模式时使用外部 SCC 模块,功耗约 0.15uA
            SETB    EX0                 ;使能 INT0 中断,用于唤醒 MCU
            SETB    EA
            NOP
            NOP
;           MOV     PCON,#IDL           ;MCU 进入 IDLE 模式
            MOV     PCON,#PD            ;MCU 进入掉电模式
            NOP
            NOP
            NOP
            NOP
            CLR     P3.5                ;测试端口
            JMP     $

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

sfr     VOCTRL      =   0xbb;
#define IDL             0x01        //PCON.0
#define PD              0x02        //PCON.1
sbit    P34         =   P3^4;
sbit    P35         =   P3^5;

void INT0_Isr() interrupt 0
{
    P34 = ~P34;                     //测试端口
}

void main()
{
    VOCTRL = 0x00;                  //掉电模式时使用内部 SCC 模块,功耗约 1.5uA
//  VOCTRL = 0x80;                  //掉电模式时使用外部 SCC 模块,功耗约 0.15uA
    EX0 = 1;                        //使能 INT0 中断,用于唤醒 MCU
    EA = 1;
    _nop_();
    _nop_();
    PCON = IDL;                     //MCU 进入 IDLE 模式
//  PCON = PD;                      //MCU 进入掉电模式
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    P35 = 0;

    while (1);
}
```

6.8.7 使用INT0/INT1/INT2/INT3/INT4 中断唤醒MCU

汇编代码

```
INTCLKO     DATA    8FH
EX2         EQU     10H
EX3         EQU     20H
EX4         EQU     40H

            ORG     0000H
            LJMP    MAIN

            ORG     0003H
            LJMP    INT0ISR
            ORG     0013H
            LJMP    INT1ISR
            ORG     0053H
            LJMP    INT2ISR
            ORG     005BH
            LJMP    INT3ISR
            ORG     0083H
            LJMP    INT4ISR
```

```
                ORG     0100H
INT0ISR:
                CPL     P1.0            ;测试端口
                RETI
INT1ISR:
                CPL     P1.0            ;测试端口
                RETI
INT2ISR:
                CPL     P1.0            ;测试端口
                RETI
INT3ISR:
                CPL     P1.0            ;测试端口
                RETI
INT4ISR:
                CPL     P1.0            ;测试端口
                RETI

MAIN:
                MOV     SP,#3FH

                CLR     IT0             ;使能 INT0 上升沿和下降沿中断
;               SETB    IT0             ;使能 INT0 下降沿中断
                SETB    EX0             ;使能 INT0 中断

                CLR     IT1             ;使能 INT1 上升沿和下降沿中断
;               SETB    IT1             ;使能 INT1 下降沿中断
                SETB    EX1             ;使能 INT1 中断

                MOV     INTCLKO,#EX2    ;使能 INT2 下降沿中断
                ORL     INTCLKO,#EX3    ;使能 INT3 下降沿中断
                ORL     INTCLKO,#EX4    ;使能 INT4 下降沿中断

                SETB    EA

                MOV     PCON,#02H       ;MCU 进入掉电模式
                NOP                     ;掉电模式被唤醒后,MCU 首先会执行此语句
                                        ;然后再进入中断服务程序
                NOP
                NOP
                NOP
LOOP:
                CPL     P1.1
                JMP     LOOP

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

sfr     INTCLKO     =   0x8f;
#define EX2             0x10
#define EX3             0x20
#define EX4             0x40

sbit    P10         =   P1^0;
sbit    P11         =   P1^1;
```

```
void INT0_Isr() interrupt 0
{
    P10 = !P10;                 //测试端口
}

void INT1_Isr() interrupt 2
{
    P10 = !P10;                 //测试端口
}

void INT2_Isr() interrupt 10
{
    P10 = !P10;                 //测试端口
}

void INT3_Isr() interrupt 11
{
    P10 = !P10;                 //测试端口
}

void INT4_Isr() interrupt 16
{
    P10 = !P10;                 //测试端口
}

void main()
{
    IT0 = 0;                    //使能 INT0 上升沿和下降沿中断
//  IT0 = 1;                    //使能 INT0 下降沿中断
    EX0 = 1;                    //使能 INT0 中断

    IT1 = 0;                    //使能 INT1 上升沿和下降沿中断
//  IT1 = 1;                    //使能 INT1 下降沿中断
    EX1 = 1;                    //使能 INT1 中断

    INTCLKO = EX2;              //使能 INT2 下降沿中断
    INTCLKO |= EX3;             //使能 INT3 下降沿中断
    INTCLKO |= EX4;             //使能 INT4 下降沿中断

    EA = 1;

    PCON = 0x02;                //MCU 进入掉电模式
    _nop_();                    //掉电模式被唤醒后,MCU 首先会执行此语句
                                //然后再进入中断服务程序
    _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}
```

6.8.8 使用T0/T1/T2/T3/T4 中断唤醒MCU

汇编代码

```
;测试工作频率为 11.0592MHz

T2L         DATA    0D7H
```

```
T2H         DATA    0D6H
T3L         DATA    0D5H
T3H         DATA    0D4H
T4L         DATA    0D3H
T4H         DATA    0D2H
T4T3M       DATA    0D1H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
ET3         EQU     20H
ET4         EQU     40H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H
T3IF        EQU     02H
T4IF        EQU     04H

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR
            ORG     001BH
            LJMP    TM1ISR
            ORG     0063H
            LJMP    TM2ISR
            ORG     009BH
            LJMP    TM3ISR
            ORG     00A3H
            LJMP    TM4ISR

            ORG     0100H
TM0ISR:
            CPL     P1.0                ;测试端口
            RETI
TM1ISR:
            CPL     P1.0                ;测试端口
            RETI
TM2ISR:
            CPL     P1.0                ;测试端口
            ANL     AUXINTIF,#NOT T2IF  ;清中断标志
            RETI
TM3ISR:
            CPL     P1.0                ;测试端口
            ANL     AUXINTIF,#NOT T3IF  ;清中断标志
            RETI
TM4ISR:
            CPL     P1.0                ;测试端口
            ANL     AUXINTIF,#NOT T4IF  ;清中断标志
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     TMOD,#00H
            MOV     TL0,#66H            ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0                 ;启动定时器
            SETB    ET0                 ;使能定时器中断

            MOV     TL1,#66H            ;65536-11.0592M/12/1000
```

```
        MOV     TH1,#0FCH
        SETB    TR1             ;启动定时器
        SETB    ET1             ;使能定时器中断

        MOV     T2L,#66H        ;65536-11.0592M/12/1000
        MOV     T2H,#0FCH
        MOV     AUXR,#10H       ;启动定时器
        MOV     IE2,#ET2        ;使能定时器中断

        MOV     T3L,#66H        ;65536-11.0592M/12/1000
        MOV     T3H,#0FCH
        MOV     T4T3M,#08H      ;启动定时器
        ORL     IE2,#ET3        ;使能定时器中断

        MOV     T4L,#66H        ;65536-11.0592M/12/1000
        MOV     T4H,#0FCH
        ORL     T4T3M,#80H      ;启动定时器
        ORL     IE2,#ET4        ;使能定时器中断

        SETB    EA

        MOV     PCON,#02H       ;MCU 进入掉电模式
        NOP                     ;掉电唤醒后不会立即进入中断服务程序,
                                ;而是等到定时器溢出后才会进入中断服务程序
        NOP
        NOP
        NOP
LOOP:
        CPL     P1.1
        JMP     LOOP

        END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为 11.0592MHz

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T4T3M   =   0xd1;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2     0x04
#define ET3     0x20
#define ET4     0x40
sfr     AUXINTIF=   0xef;
#define T2IF    0x01
#define T3IF    0x02
#define T4IF    0x04

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;
```

```
void TM0_Isr() interrupt 1
{
    P10 = !P10;                 //测试端口
}

void TM1_Isr() interrupt 3
{
    P10 = !P10;                 //测试端口
}

void TM2_Isr() interrupt 12
{
    P10 = !P10;                 //测试端口
    AUXINTIF &= ~T2IF;          //清中断标志
}

void TM3_Isr() interrupt 19
{
    P10 = !P10;                 //测试端口
    AUXINTIF &= ~T3IF;          //清中断标志
}

void TM4_Isr() interrupt 20
{
    P10 = !P10;                 //测试端口
    AUXINTIF &= ~T4IF;          //清中断标志
}

void main()
{
    TMOD = 0x00;
    TL0 = 0x66;                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                    //启动定时器
    ET0 = 1;                    //使能定时器中断

    TL1 = 0x66;                 //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                    //启动定时器
    ET1 = 1;                    //使能定时器中断

    T2L = 0x66;                 //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                //启动定时器
    IE2 = ET2;                  //使能定时器中断

    T3L = 0x66;                 //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x08;               //启动定时器
    IE2 |= ET3;                 //使能定时器中断

    T4L = 0x66;                 //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M |= 0x80;              //启动定时器
    IE2 |= ET4;                 //使能定时器中断

    EA = 1;
```

STC MCU

```
    PCON = 0x02;                    //MCU 进入掉电模式
    _nop_();                        //掉电唤醒后不会立即进入中断服务程序,
                                    //而是等到定时器溢出后才会进入中断服务程序
    _nop_();
    _nop_();
    _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}
```

6.8.9 使用RxD/RxD2/RxD3/RxD4 中断唤醒MCU

汇编代码

```
;测试工作频率为 11.0592MHz

IE2         DATA    0AFH
ES2         EQU     01H
ES3         EQU     08H
ES4         EQU     10H

P_SW1       DATA    0A2H
P_SW2       DATA    0BAH


            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART1ISR
            ORG     0043H
            LJMP    UART2ISR
            ORG     008BH
            LJMP    UART3ISR
            ORG     0093H
            LJMP    UART4ISR

            ORG     0100H
UART1ISR:
            RETI
UART2ISR:
            RETI
UART3ISR:
            RETI
UART4ISR:
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     P_SW1,#00H          ;RXD/P3.0 下降沿唤醒
;           MOV     P_SW1,#40H          ;RXD_2/P3.6 下降沿唤醒
;           MOV     P_SW1,#80H          ;RXD_3/P1.6 下降沿唤醒
;           MOV     P_SW1,#0C0H         ;RXD_4/P4.3 下降沿唤醒
```

```
                MOV     P_SW2,#00H              ;RXD2/P1.0 下降沿唤醒
;               MOV     P_SW2,#01H              ;RXD2_2/P4.0 下降沿唤醒

                MOV     P_SW2,#00H              ;RXD3/P0.0 下降沿唤醒
;               MOV     P_SW2,#02H              ;RXD3_2/P5.0 下降沿唤醒

                MOV     P_SW2,#00H              ;RXD4/P0.2 下降沿唤醒
;               MOV     P_SW2,#04H              ;RXD4_2/P5.2 下降沿唤醒

                SETB    ES                      ;使能串口中断
                MOV     IE2,#ES2                ;使能串口中断
                ORL     IE2,#ES3                ;使能串口中断
                ORL     IE2,#ES4                ;使能串口中断
                SETB    EA

                MOV     PCON,#02H               ;MCU 进入掉电模式
                NOP                             ;掉电唤醒后不会进入中断服务程序,
                NOP
                NOP
                NOP
LOOP:
                CPL     P1.1
                JMP     LOOP

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为 11.0592MHz

sfr     IE2     =   0xaf;
#define ES2         0x01
#define ES3         0x08
#define ES4         0x10

sfr     P_SW1   =   0xa2;
sfr     P_SW2   =   0xba;

sbit    P11     =   P1^1;

void UART1_Isr() interrupt 4
{
}

void UART2_Isr() interrupt 8
{
}

void UART3_Isr() interrupt 17
{
}

void UART4_Isr() interrupt 18
{
}
```

```
void main()
{
    P_SW1 = 0x00;                   //RXD/P3.0 下降沿唤醒
//  P_SW1 = 0x40;                   //RXD_2/P3.6 下降沿唤醒
//  P_SW1 = 0x80;                   //RXD_3/P1.6 下降沿唤醒
//  P_SW1 = 0xc0;                   //RXD_4/P4.3 下降沿唤醒

    P_SW2 = 0x00;                   //RXD2/P1.0 下降沿唤醒
//  P_SW2 = 0x01;                   //RXD2_2/P4.0 下降沿唤醒

    P_SW2 = 0x00;                   // RXD3/P0.0 下降沿唤醒
//  P_SW2 = 0x02;                   // RXD3_2/P5.0 下降沿唤醒

    P_SW2 = 0x00;                   //RXD4/P0.2 下降沿唤醒
//  P_SW2 = 0x04;                   //RXD4_2/P5.2 下降沿唤醒

    ES = 1;                         //使能串口中断
    IE2 = ES2;                      //使能串口中断
    IE2 |= ES3;                     //使能串口中断
    IE2 |= ES4;                     //使能串口中断
    EA = 1;

    PCON = 0x02;                    //MCU 进入掉电模式
    _nop_();                        //掉电唤醒后不会进入中断服务程序,
    _nop_();
    _nop_();
    _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}
```

6.8.10 使用LVD中断唤醒MCU

汇编代码

```
RSTCFG      DATA    0FFH
ENLVR       EQU     40H             ;RSTCFG.6
LVD2V2      EQU     00H             ;LVD@2.2V
LVD2V4      EQU     01H             ;LVD@2.4V
LVD2V7      EQU     02H             ;LVD@2.7V
LVD3V0      EQU     03H             ;LVD@3.0V
ELVD        BIT     IE.6
LVDF        EQU     20H             ;PCON.5

            ORG     0000H
            LJMP    MAIN
            ORG     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
            ANL     PCON,#NOT LVDF  ;清中断标志
            CPL     P1.0            ;测试端口
            RETI

MAIN:
```

```
        MOV     SP,#3FH

        ANL     PCON,#NOT LVDF      ;上电需要清中断标志
        MOV     RSTCFG,# LVD3V0     ;设置 LVD 电压为 3.0V
        SETB    ELVD                ;使能 LVD 中断
        SETB    EA

        MOV     PCON,#02H           ;MCU 进入掉电模式
        NOP                         ;掉电唤醒后立即进入中断服务程序
        NOP
        NOP
        NOP
LOOP:
        CPL     P1.1
        JMP     LOOP

        END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

sfr     RSTCFG      =   0xff;
#define ENLVR           0x40    //RSTCFG.6
#define LVD2V2          0x00    //LVD@2.2V
#define LVD2V4          0x01    //LVD@2.4V
#define LVD2V7          0x02    //LVD@2.7V
#define LVD3V0          0x03    //LVD@3.0V
sbit    ELVD        =   IE^6;
#define LVDF            0x20    //PCON.5

sbit    P10         =   P1^0;
sbit    P11         =   P1^1;

void LVD_Isr() interrupt 6
{
    PCON &= ~LVDF;              //清中断标志
    P10 = !P10;                 //测试端口
}

void main()
{
    PCON &= ~LVDF;              //上电需要清中断标志
    RSTCFG = LVD3V0;            //设置 LVD 电压为 3.0V
    ELVD = 1;                   //使能 LVD 中断
    EA = 1;

    PCON = 0x02;                //MCU 进入掉电模式
    _nop_();                    //掉电唤醒后立即进入中断服务程序
    _nop_();
    _nop_();
    _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}
```

STC MCU

6.8.11 使用CCP0/CCP1/CCP2/CCP3 中断唤醒MCU

汇编代码

```
;测试工作频率为 11.0592MHz

CCON        DATA    0D8H
CF          BIT     CCON.7
CR          BIT     CCON.6
CCF3        BIT     CCON.3
CCF2        BIT     CCON.2
CCF1        BIT     CCON.1
CCF0        BIT     CCON.0
CMOD        DATA    0D9H
CL          DATA    0E9H
CH          DATA    0F9H
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H
CCAPM3      DATA    0DDH
CCAP3L      DATA    0EDH
CCAP3H      DATA    0FDH
PCA_PWM3    DATA    0F5H

P_SW1       DATA    0A2H

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            ANL     CCON,#NOT 8FH       ;清中断标志
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     P_SW1,#00H      ; CCP0/P1.7, CCP1/P1.6, CCP2/P1.5,CCP3/P1.4
;           MOV     P_SW1,#10H      ; CCP0_2/P2.3, CCP1_2/P2.4, CCP2_2/P2.5,CCP3_2/P2.6
;           MOV     P_SW1,#20H      ; CCP0_3/P7.0, CCP1_3/P7.1, CCP2_3/P7.2,CCP3_3/P7.3
;           MOV     P_SW1,#30H      ; CCP0_4/P3.3, CCP1_4/P3.2, CCP2_4/P3.1,CCP3_4/P3.0

            MOV     CCON,#00H
            MOV     CMOD,#08H           ;PCA 时钟为系统时钟
            MOV     CCAPM0,#31H         ;使能 CCP0 口边沿唤醒功能
            MOV     CCAPM1,#31H         ;使能 CCP1 口边沿唤醒功能
```

```
                MOV     CCAPM2,#31H         ;使能 CCP2 口边沿唤醒功能
                MOV     CCAPM3,#31H         ;使能 CCP3 口边沿唤醒功能
                SETB    CR                  ;启动 PCA 计时器
                SETB    EA

                MOV     PCON,#02H           ;MCU 进入掉电模式
                NOP                         ;掉电唤醒后立即进入中断服务程序
                NOP
                NOP
                NOP

LOOP:
                CPL     P1.1
                JMP     LOOP

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为 11.0592MHz

sfr     CCON    =   0xd8;
sbit    CF      =   CCON^7;
sbit    CR      =   CCON^6;
sbit    CCF3    =   CCON^3;
sbit    CCF2    =   CCON^2;
sbit    CCF1    =   CCON^1;
sbit    CCF0    =   CCON^0;
sfr     CMOD    =   0xd9;
sfr     CL      =   0xe9;
sfr     CH      =   0xf9;
sfr     CCAPM0  =   0xda;
sfr     CCAP0L  =   0xea;
sfr     CCAP0H  =   0xfa;
sfr     PCA_PWM0=   0xf2;
sfr     CCAPM1  =   0xdb;
sfr     CCAP1L  =   0xeb;
sfr     CCAP1H  =   0xfb;
sfr     PCA_PWM1=   0xf3;
sfr     CCAPM2  =   0xdc;
sfr     CCAP2L  =   0xec;
sfr     CCAP2H  =   0xfc;
sfr     PCA_PWM2=   0xf4;
sfr     CCAPM3  =   0xdd;
sfr     CCAP3L  =   0xed;
sfr     CCAP3H  =   0xfd;
sfr     PCA_PWM3=   0xf5;

sfr     P_SW1   =   0xa2;

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

void PCA_Isr() interrupt 7
{
    CCON &= ~0x8f;                  //清中断标志
```

```
    P10 = !P10;                     //测试端口
}

void main()
{
    P_SW1 = 0x00;                   //CCP0/P1.7, CCP1/P1.6, CCP2/P1.5,CCP3/P1.4
//  P_SW1 = 0x10;                   //CCP0_2/P2.3, CCP1_2/P2.4, CCP2_2/P2.5,CCP3_2/P2.6
//  P_SW1 = 0x20;                   //CCP0_3/P7.0, CCP1_3/P7.1, CCP2_3/P7.2,CCP3_3/P7.3
//  P_SW1 = 0x30;                   //CCP0_4/P3.3, CCP1_4/P3.2, CCP2_4/P3.1,CCP3_4/P3.0

    CCON = 0x00;
    CMOD = 0x08;                    //PCA 时钟为系统时钟
    CCAPM0 = 0x31;                  //使能 CCP0 口边沿唤醒功能
    CCAPM1 = 0x31;                  //使能 CCP1 口边沿唤醒功能
    CCAPM2 = 0x31;                  //使能 CCP2 口边沿唤醒功能
    CCAPM3 = 0x31;                  //使能 CCP3 口边沿唤醒功能

    CR = 1;                         //启动 PCA 计时器
    EA = 1;

    PCON = 0x02;                    //MCU 进入掉电模式
    _nop_();                        //掉电唤醒后立即进入中断服务程序
    _nop_();
    _nop_();
    _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}
```

6.8.12 CMP中断唤醒MCU

汇编代码

```
CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

            ORG     0000H
            LJMP    MAIN
            ORG     00ABH
            LJMP    CMPISR

            ORG     0100H
CMPISR:
            ANL     CMPCR1,#NOT 40H     ;清中断标志
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     CMPCR2,#00H
            MOV     CMPCR1,#80H         ;使能比较器模块
            ORL     CMPCR1,#30H         ;使能比较器边沿中断
            ANL     CMPCR1,#NOT 08H     ;P3.6 为CMP+输入脚
```

```
                ORL     CMPCR1,#04H             ;P3.7 为CMP-输入脚
                ORL     CMPCR1,#02H             ;使能比较器输出
                SETB    EA

                MOV     PCON,#02H               ;MCU 进入掉电模式
                NOP                             ;掉电唤醒后立即进入中断服务程序
                NOP
                NOP
                NOP

LOOP:
                CPL     P1.1
                JMP     LOOP

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

sfr     CMPCR1  =   0xe6;
sfr     CMPCR2  =   0xe7;

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

void CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;                //清中断标志
    P10 = !P10;                     //测试端口
}

void main()
{
    CMPCR2 = 0x00;
    CMPCR1 = 0x80;                  //使能比较器模块
    CMPCR1 |= 0x30;                 //使能比较器边沿中断
    CMPCR1 &= ~0x08;                //P3.6 为CMP+输入脚
    CMPCR1 |= 0x04;                 //P3.7 为CMP-输入脚
    CMPCR1 |= 0x02;                 //使能比较器输出
    EA = 1;

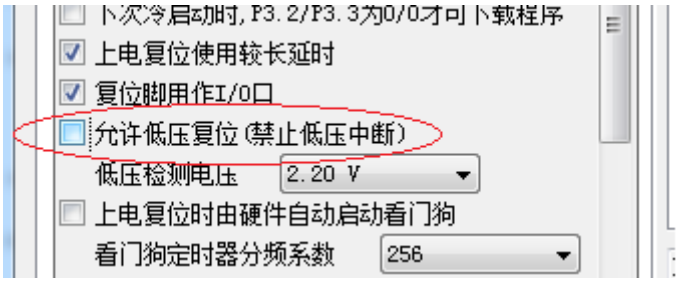
    PCON = 0x02;                    //MCU 进入掉电模式
    _nop_();                        //掉电唤醒后立即进入中断服务程序
    _nop_();
    _nop_();
    _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}
```

6.8.13 使用LVD功能检测工作电压（电池电压）

若需要使用 LVD 功能检测电池电压，则在 ISP 下载时需要将低压复位功能去掉，如下图 “允许低压复

位（禁止低压中断）”的硬件选项的勾选项需要去掉



汇编代码

```
RSTCFG      DATA    0FFH
LVD2V2      EQU     00H         ;LVD@2.2V
LVD2V4      EQU     01H         ;LVD@2.4V
LVD2V7      EQU     02H         ;LVD@2.7V
LVD3V0      EQU     03H         ;LVD@3.0V
LVDF        EQU     20H         ;PCON.5

            ORG     0000H
            JMP     MAIN

            ORG     0100H
MAIN:
            ANL     PCON,#NOT LVDF
            MOV     RSTCFG,#LVD3V0
LOOP:
            MOV     B,#0FH

            MOV     RSTCFG,#LVD3V0
            CALL    DELAY
            ANL     PCON,#NOT LVDF
            CALL    DELAY
            MOV     A,PCON
            ANL     A,#LVDF
            JZ      SKIP
            MOV     A,B
            CLR     C
            RRC     A
            MOV     B,A

            MOV     RSTCFG,#LVD2V7
            CALL    DELAY
            ANL     PCON,#NOT LVDF
            CALL    DELAY
            MOV     A,PCON
            ANL     A,#LVDF
            JZ      SKIP
            MOV     A,B
            CLR     C
            RRC     A
            MOV     B,A

            MOV     RSTCFG,#LVD2V4
            CALL    DELAY
            ANL     PCON,#NOT LVDF
            CALL    DELAY
            MOV     A,PCON
```

```
                ANL     A,#LVDF
                JZ      SKIP
                MOV     A,B
                CLR     C
                RRC     A
                MOV     B,A

                MOV     RSTCFG,#LVD2V2
                CALL    DELAY
                ANL     PCON,#NOT LVDF
                CALL    DELAY
                MOV     A,PCON
                ANL     A,#LVDF
                JZ      SKIP
                MOV     A,B
                CLR     C
                RRC     A
                MOV     B,A

SKIP:
                MOV     A,B
                CPL     A
                MOV     P2,A                ;P2.3~P2.0 显示电池电量
                JMP     LOOP

DELAY:
                MOV     R0,#100
NEXT:
                NOP
                NOP
                NOP
                NOP
                DJNZ    R0,NEXT
                RET

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

#define FOSC            24000000UL
#define T1MS            (65536 - FOSC/4/100)

sfr     RSTCFG      =   0xff;
#define LVD2V2          0x00        //LVD@2.2V
#define LVD2V4          0x01        //LVD@2.4V
#define LVD2V7          0x02        //LVD@2.7V
#define LVD3V0          0x03        //LVD@3.0V

#define LVDF            0x20        //PCON.5

void delay()
{
    int i;

    for (i=0; i<100; i++)
    {
```

```
            _nop_();
            _nop_();
            _nop_();
            _nop_();
        }
}

void main()
{
        unsigned char power;

        PCON &= ~LVDF;
        RSTCFG = LVD3V0;
        while (1)
        {
            power = 0x0f;

            RSTCFG = LVD3V0;
            delay();
            PCON &= ~LVDF;
            delay();
            if (PCON & LVDF)
            {
                power >>= 1;

                RSTCFG = LVD2V7;
                delay();
                PCON &= ~LVDF;
                delay();
                if (PCON & LVDF)
                {
                    power >>= 1;

                    RSTCFG = LVD2V4;
                    delay();
                    PCON &= ~LVDF;
                    delay();
                    if (PCON & LVDF)
                    {
                        power >>= 1;

                        RSTCFG = LVD2V2;
                        delay();
                        PCON &= ~LVDF;
                        delay();
                        if (PCON & LVDF)
                        {
                            power >>= 1;
                        }
                    }
                }
            }
            RSTCFG = LVD3V0;
            P2 = ~power;                    //P2.3~P2.0 显示电池电量
        }
}
```

7 存储器

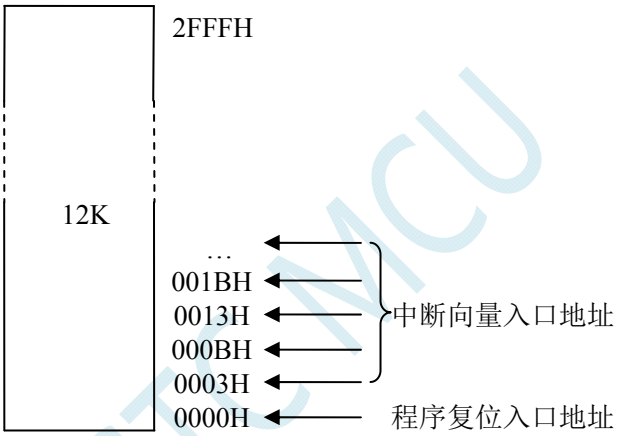
STC8A/8F 系列单片机的程序存储器和数据存储器是各自独立编址的。由于没有提供访问外部程序存储器的总线，单片机的所有程序存储器都是片上 Flash 存储器，不能访问外部程序存储器。

STC8A/8F 系列单片机内部集成了大容量的数据存储器。STC8A/8F 系列单片机内部的数据存储器在物理和逻辑上都分为两个地址空间:内部 RAM(256 字节)和内部扩展 RAM。其中内部 RAM 的高 128 字节的数据存储器与特殊功能寄存器(SFRs)地址重叠，实际使用时通过不同的寻址方式加以区分。

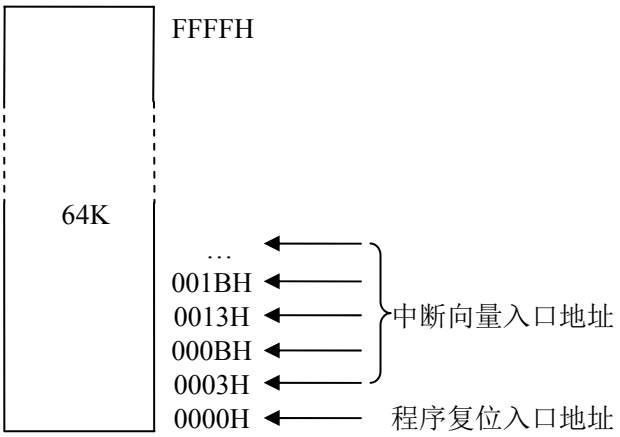
7.1 程序存储器

程序存储器用于存放用户程序、数据以及表格等信息。

STC8F1K08S2-20Pin/16Pin 系列、STC8F1K08-8Pin 系列单片内部集成了 12K 字节的 Flash 程序存储器（ROM）。



STC8A8K64S4A12-64Pin 系列、STC8A4K64S2A12-64Pin 系列、STC8F2K64S4-44Pin 系列和 STC8F2K64S2-44Pin 系列单片机内部集成了 64K 字节的 Flash 程序存储器（ROM）。



单片机复位后，程序计数器(PC)的内容为 0000H，从 0000H 单元开始执行程序。另外中断服务程序的入口地址(又称中断向量)也位于程序存储器单元。在程序存储器中，每个中断都有一个固定的入口地址，当中断发生并得到响应后，单片机就会自动跳转到相应的中断入口地址去执行程序。外部中断 0（INT0）的中断服务程序的入口地址是 0003H，定时器/计数器 0（TIMER0）中断服务程序的入口地址是 000BH，外部中断 1（INT1）的中断服务程序的入口地址是 0013H，定时器/计数器 1（TIMER1）的中断服务程序的入口地址是 001BH 等。更多的中断服务程序的入口地址(中断向量)请参考中断介绍章节。

由于相邻中断入口地址的间隔区间仅仅有 8 个字节，一般情况下无法保存完整的中断服务程序，因此在中断响应的地址区域存放一条无条件转移指令，指向真正存放中断服务程序的空间去执行。

STC8A/8F 系列单片机中都包含有 Flash 数据存储器（EEPROM）。以字节为单位进行读/写数据，以 512 字节为页单位进行擦除，可在线反复编程擦写 10 万次以上，提高了使用的灵活性和方便性。

STC MCU

7.2 数据存储器

STC8A/8F 系列单片机内部集成的 RAM 可用于存放程序执行的中间结果和过程数据。

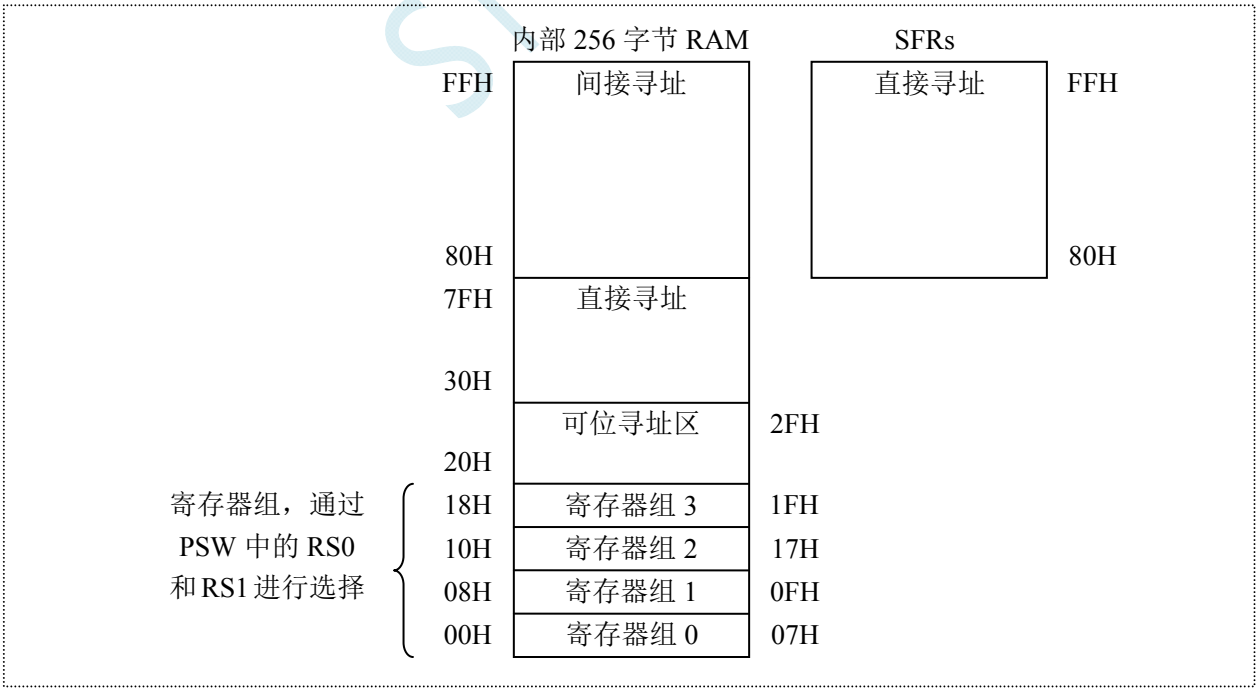
单片机系列	内部直接访问 RAM（DATA）	内部直接访问 RAM（IDATA）	内部扩展 RAM（XDATA）
STC8A8K64S4A12-64Pin 系列	128 字节	128 字节	8192 字节
STC8A4K64S2A12-64Pin 系列	128 字节	128 字节	4096 字节
STC8F2K64S4-44Pin 系列	128 字节	128 字节	2048 字节
STC8F2K64S2-44Pin 系列	128 字节	128 字节	2048 字节
STC8F1K08S2-20Pin/16Pin 系列	128 字节	128 字节	1024 字节
STC8F1K08-8Pin 系列	128 字节	128 字节	1024 字节

此外，STC8A/8F 系列封装管脚数为 40 及其以上的单片机还可以访问在片外扩展的 64KB 外部数据存储器。

7.2.1 内部RAM

内部 RAM 共 256 字节，可分为 2 个部分：低 128 字节 RAM 和高 128 字节 RAM。低 128 字节的数据存储器与传统 8051 兼容，既可直接寻址也可间接寻址。高 128 字节 RAM（在 8052 中扩展了高 128 字节 RAM）与特殊功能寄存器区共用相同的逻辑地址，都使用 80H~FFH，但在物理上是分别独立的，使用时通过不同的寻址方式加以区分。高 128 字节 RAM 只能间接寻址，特殊功能寄存器区只可直接寻址。

内部 RAM 的结构如下图所示：



低 128 字节 RAM 也称通用 RAM 区。通用 RAM 区又可分为工作寄存器组区，可位寻址区，用户 RAM 区和堆栈区。工作寄存器组区地址从 00H~1FH 共 32 字节单元，分为 4 组，每一组称为一个寄存器组，每组

包含 8 个 8 位的工作寄存器，编号均为 R0～R7，但属于不同的物理空间。通过使用工作寄存器组，可以提高运算速度。R0~R7 是常用的寄存器，提供 4 组是因为 1 组往往不够用。程序状态字 PSW 寄存器中的 RS1 和 RS0 组合决定当前使用的工作寄存器组，见下面 PSW 寄存器的介绍。

7.2.2 程序状态寄存器（PSW）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PSW	D0H	CY	AC	F0	**RS1**	**RS0**	OV	F1	P

CY：进/借位标志位。

AC：辅组进/借位标志位。

F0：用户标志位 0。

RS1，RS0：工作寄存器选择位

RS1	RS0	工作寄存器组（R0~R7）
0	0	第 0 组（00H～07H）
0	1	第 1 组（08H～0FH）
1	0	第 2 组（10H～17H）
1	1	第 3 组（18H～1FH）

OV：溢出标志位。

F1：用户标志位 1。

P：奇偶校验标志位。

可位寻址区的地址从 20H～2FH 共 16 个字节单元。20H~2FH 单元既可像普通 RAM 单元一样按字节存取，也可以对单元中的任何一位单独存取，共 128 位，所对应的逻辑位地址范围是 00H~7FH。位地址范围是 00H~7FH，内部 RAM 低 128 字节的地址也是 00H~7FH，从外表看，二者地址是一样的，实际上二者具有本质的区别；位地址指向的是一个位，而字节地址指向的是一个字节单元，在程序中使用不同的指令区分。

内部 RAM 中的 30H~FFH 单元是用户 RAM 和堆栈区。一个 8 位的堆栈指针(SP)，用于指向堆栈区。单片机复位后，堆栈指针 SP 为 07H，指向了工作寄存器组 0 中的 R7，因此，用户初始化程序都应对 SP 设置初值，一般设置在 80H 以后的单元为宜。

堆栈指针是一个 8 位专用寄存器。它指示出堆栈顶部在内部 RAM 块中的位置。系统复位后，SP 初始化位 07H，使得堆栈事实上由 08H 单元开始，考虑 08H~1FH 单元分别属于工作寄存器组 1~3，若在程序设计中用到这些区，则最好把 SP 值改变为 80H 或更大的值为宜。STC8 系列单片机的堆栈是向上生长的，即将数据压入堆栈后，SP 内容增大。

7.2.3 内部扩展RAM，XRAM，XDATA

STC8A/8F 系列单片机片内除了集成 256 字节的内部 RAM 外，还集成了内部的扩展 RAM。访问内部扩展 RAM 的方法和传统 8051 单片机访问外部扩展 RAM 的方法相同，但是不影响 P0 口(数据总线和高八位地址总线)、P2 口(低八位地址总线)、以及 RD、WR 和 ALE 等端口上的信号。

在汇编语言中，内部扩展 RAM 通过 MOVX 指令访问，

```
MOVX    A,@DPTR
MOVX    @DPTR,A
MOVX    A,@Ri
MOVX    @Ri,A
```

在 C 语言中，可使用 xdata/pdata 声明存储类型即可。如：

```
unsigned char xdata i;
unsigned int pdata j;
```

注：pdata 即为 xdata 的低 256 字节，在 C 语言中定义变量为 pdata 类型后，编译器会自动将变量分配在 XDATA 的 0000H~00FFH 区域，并使用 MOVX @Ri,A 和 MOVX A@Ri 进行访问。

单片机内部扩展 RAM 是否可以访问，受辅助寄存器 AUXR 中的 EXTRAM 位控制。

7.2.4 辅助寄存器（AUXR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	**EXTRAM**	S1ST2

EXTRAM：扩展 RAM 访问控制

0：访问内部扩展 RAM。

1：内部扩展 RAM 被禁用。

STC MCU

7.2.5 外部扩展RAM，XRAM，XDATA

STC8A/8F 系列封装管脚数为 40 及其以上的单片机具有扩展 64KB 外部数据存储器的能力。访问外部数据存储器期间，WR/RD/ALE 信号要有效。STC8A/8F 系列单片机新增了一个控制外部 64K 字节数据总线速度的特殊功能寄存器 BUS_SPEED，说明如下：

7.2.6 总线速度控制寄存器（BUS_SPEED）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
BUS_SPEED	A1H	RW_S[1:0]					SPEED[2:0]		

RW_S[1:0]：RD/WR 控制线选择位

00：P4.4 为 RD，P4.3 为 WR

01：P3.7 为 RD，P3.6 为 WR

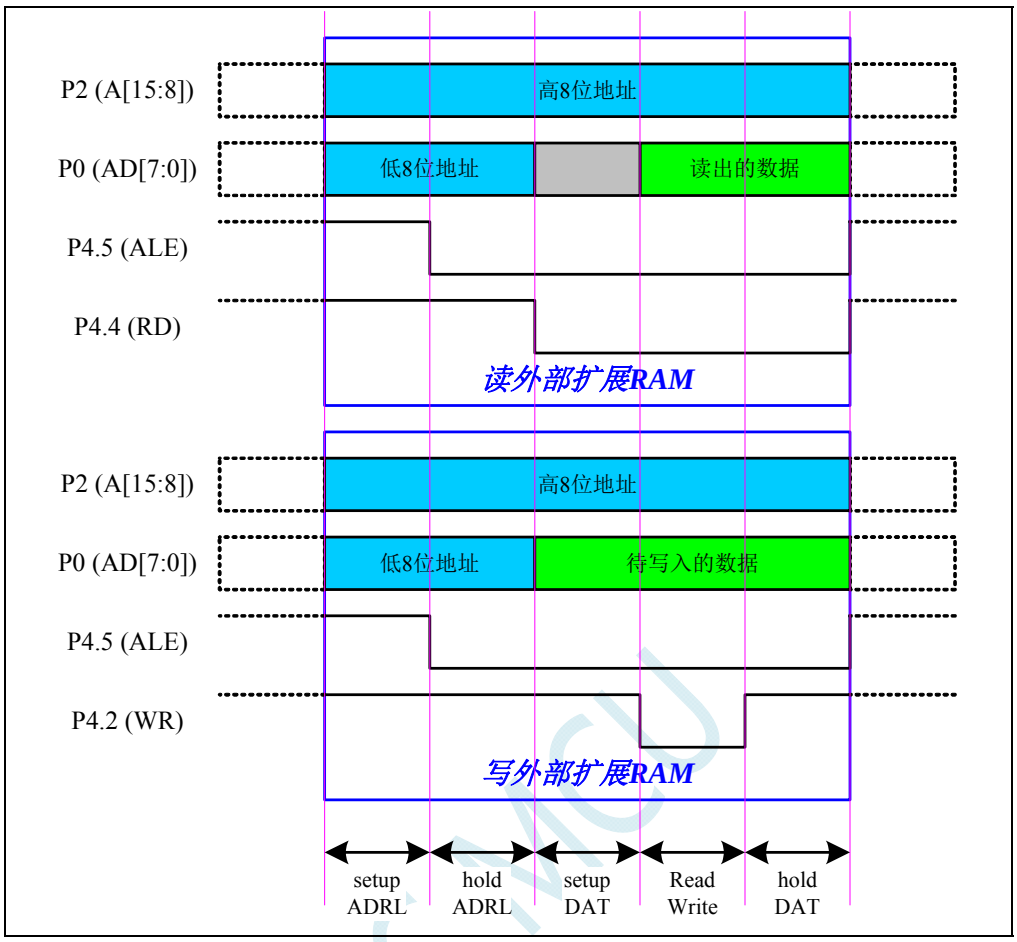
10：P4.2 为 RD，P4.0 为 WR

11：保留

SPEED[2:0]：总线读写速度控制（读写数据时控制信号和数据信号的准备时间和保持时间）

指令	时钟数	
	访问内部扩展 RAM	访问外部扩展 RAM
MOVX A,@Ri	3	3+5* (SPEED+1)
MOVX @Ri,A	3	3+5* (SPEED+1)
MOVX A,@DPTR	2	2+5* (SPEED+1)
MOVX @DPTR,A	2	2+5* (SPEED+1)

读写外部扩展 RAM 时序如下图所示：



7.2.7 8051 中可位寻址的数据存储器

8051 单片机内部可位寻址的数据存储器包括两部分：第一部分的地址范围为 00H~7FH，第二部分的地址范围是 80H~FFH。00H~7FH 的位寻址区域是数据区 20H~2FH 这 16 个字节的映射；而 80H~FFh 的位寻址区域则是所有的特殊功能寄存器中地址能被 8 整除的 16 个特殊功能寄存器（包括 80H、88H、90H、98H、A0H、A8H、B0H、B8H、C0H、C8H、D0H、D8H、E0H、E8H、F0H、F8H）的映射。

数据存储器地址	位寻址地址							
	B7	B6	B5	B4	B3	B2	B1	B0
F8H（P7）	FFH F8H.7	FEH F8H.6	FDH F8H.5	FCH F8H.4	FBH F8H.3	FAH F8H.2	F9H F8H.1	F8H F8H.0
F0H（B）	F7H F0H.7	F6H F0H.6	F5H F0H.5	F4H F0H.4	F3H F0H.3	F2H F0H.2	F1H F0H.1	F0H F0H.0
E8H（P6）	EFH E8H.7	EEH E8H.6	EDH E8H.5	ECH E8H.4	EBH E8H.3	EAH E8H.2	E9H E8H.1	E8H E8H.0
E0H（ACC）	E7H E0H.7	E6H E0H.6	E5H E0H.5	E4H E0H.4	E3H E0H.3	E2H E0H.2	E1H E0H.1	E0H E0H.0
D8H（CCON）	DFH D8H.7	DEH D8H.6	DDH D8H.5	DCH D8H.4	DBH D8H.3	DAH D8H.2	D9H D8H.1	D8H D8H.0
D0H（PSW）	D7H D0H.7	D6H D0H.6	D5H D0H.5	D4H D0H.4	D3H D0H.3	D2H D0H.2	D1H D0H.1	D0H D0H.0
C8H（P5）	CFH C8H.7	CEH C8H.6	CDH C8H.5	CCH C8H.4	CBH C8H.3	CAH C8H.2	C9H C8H.1	C8H C8H.0
C0H（P4）	C7H C0H.7	C6H C0H.6	C5H C0H.5	C4H C0H.4	C3H C0H.3	C2H C0H.2	C1H C0H.1	C0H C0H.0
B8H（IP）	BFH B8H.7	BEH B8H.6	BDH B8H.5	BCH B8H.4	BBH B8H.3	BAH B8H.2	B9H B8H.1	B8H B8H.0
B0H（P3）	B7H B0H.7	B6H B0H.6	B5H B0H.5	B4H B0H.4	B3H B0H.3	B2H B0H.2	B1H B0H.1	B0H B0H.0
A8H（IE）	AFH A8H.7	AEH A8H.6	ADH A8H.5	ACH A8H.4	ABH A8H.3	AAH A8H.2	A9H A8H.1	A8H A8H.0
A0H（P2）	A7H A0H.7	A6H A0H.6	A5H A0H.5	A4H A0H.4	A3H A0H.3	A2H A0H.2	A1H A0H.1	A0H A0H.0
98H（SCON）	9FH 98H.7	9EH 98H.6	9DH 98H.5	9CH 98H.4	9BH 98H.3	9AH 98H.2	99H 98H.1	98H 98H.0
90H（P1）	97H 90H.7	96H 90H.6	95H 90H.5	94H 90H.4	93H 90H.3	92H 90H.2	91H 90H.1	90H 90H.0
88H（TCON）	8FH 88H.7	8EH 88H.6	8DH 88H.5	8CH 88H.4	8BH 88H.3	8AH 88H.2	89H 88H.1	88H 88H.0
80H（P0）	87H 80H.7	86H 80H.6	85H 80H.5	84H 80H.4	83H 80H.3	82H 80H.2	81H 80H.1	80H 80H.0
2FH	7FH 2FH.7	7EH 2FH.6	7DH 2FH.5	7CH 2FH.4	7BH 2FH.3	7AH 2FH.2	79H 2FH.1	78H 2FH.0
2EH	77H 2EH.7	76H 2EH.6	75H 2EH.5	74H 2EH.4	73H 2EH.3	72H 2EH.2	71H 2EH.1	70H 2EH.0
2DH	6FH 2DH.7	6EH 2DH.6	6DH 2DH.5	6CH 2DH.4	6BH 2DH.3	6AH 2DH.2	69H 2DH.1	68H 2DH.0
2CH	67H 2CH.7	66H 2CH.6	65H 2CH.5	64H 2CH.4	63H 2CH.3	62H 2CH.2	61H 2CH.1	60H 2CH.0
2BH	5FH 2BH.7	5EH 2BH.6	5DH 2BH.5	5CH 2BH.4	5BH 2BH.3	5AH 2BH.2	59H 2BH.1	58H 2BH.0
2AH	57H 2AH.7	56H 2AH.6	55H 2AH.5	54H 2AH.4	53H 2AH.3	52H 2AH.2	51H 2AH.1	50H 2AH.0

29H	4FH 29H.7	4EH 29H.6	4DH 29H.5	4CH 29H.4	4BH 29H.3	4AH 29H.2	49H 29H.1	48H 29H.0
28H	47H 28H.7	46H 28H.6	45H 28H.5	44H 28H.4	43H 28H.3	42H 28H.2	41H 28H.1	40H 28H.0
27H	3FH 27H.7	3EH 27H.6	3DH 27H.5	3CH 27H.4	3BH 27H.3	3AH 27H.2	39H 27H.1	38H 27H.0
26H	37H 26H.7	36H 26H.6	35H 26H.5	34H 26H.4	33H 26H.3	32H 26H.2	31H 26H.1	30H 26H.0
25H	2FH 25H.7	2EH 25H.6	2DH 25H.5	2CH 25H.4	2BH 25H.3	2AH 25H.2	29H 25H.1	28H 25H.0
24H	27H 24H.7	26H 24H.6	25H 24H.5	24H 24H.4	23H 24H.3	22H 24H.2	21H 24H.1	20H 24H.0
23H	1FH 23H.7	1EH 23H.6	1DH 23H.5	1CH 23H.4	1BH 23H.3	1AH 23H.2	19H 23H.1	18H 23H.0
22H	17H 22H.7	16H 22H.6	15H 22H.5	14H 22H.4	13H 22H.3	12H 22H.2	11H 22H.1	10H 22H.0
21H	0FH 21H.7	0EH 21H.6	0DH 21H.5	0CH 21H.4	0BH 21H.3	0AH 21H.2	09H 21H.1	08H 21H.0
20H	07H 20H.7	06H 20H.6	05H 20H.5	04H 20H.4	03H 20H.3	02H 20H.2	01H 20H.1	00H 20H.0

STC MCU

7.3 存储器中的特殊参数，在ISP下载时可烧录进程序FLASH

STC8G 系列单片机内部的数据存储器和程序存储器中保存有与芯片相关的一些特殊参数，包括：全球唯一 ID 号、32K 掉电唤醒定时器的频率、内部 1.344V 参考信号源值以及 IRC 参数。

这些参数在 Flash 程序存储器（ROM）中的存放地址分别如下：

参数名称	保存地址				参数说明
	STC8A8K16S4A12 STC8A4K16S4A12 STC8F2K16S4 STC8F2K16S2	**STC8A8K32S4A12 STC8A4K32S4A12 STC8F2K32S4 STC8F2K32S2**	**STC8A8K60S4A12 STC8A4K60S4A12 STC8F2K60S4 STC8F2K60S2**	**STC8A8K64S4A12 STC8A4K64S4A12 STC8F2K64S4 STC8F2K64S2**	
全球唯一 ID 号	3FF9H~3FFFH	7FF9H~7FFFH	0EFF9H~0EFFFH	0FDF9H~0FDFFH	7 字节
Bandgap 电压值	3FF7H~3FF8H	7FF7H~7FF8H	0EFF7H~0EFF8H	0FDF7H~0FDF8H	电压单位为毫伏
32K 掉电唤醒定时器的频率	3FF5H~3FF6H	7FF5H~7FF6H	0EFF5H~0EFF6H	0FDF5H~0FDF6H	单位 Hz
22.1184MHz 的 IRC 参数	3FF4H	7FF4H	0EFF4H	0FDF4H	—
24MHz 的 IRC 参数	3FF3H	7FF3H	0EFF3H	0FDF3H	—

参数名称	保存地址			参数说明
	STC8F1K08 STC8F1K08S2	**STC8F1K12 STC8F1K12S2**	**STC8F1K17 STC8F1K17S2**	
全球唯一 ID 号	1FF9H~1FFFH	2FF9H~2FFFH	43F9H~43FFH	7 字节
内部 1.344V 参考信号源	1FF7H~1FF8H	2FF7H~2FF8H	43F7H~43F8H	毫伏（高字节在前）
32K 掉电唤醒定时器的频率	1FF5H~1FF6H	2FF5H~2FF6H	43F5H~43F6H	Hz （高字节在前）
22.1184MHz 的 IRC 参数	1FF4H	2FF4H	43F4H	—
24MHz 的 IRC 参数	1FF3H	2FF3H	43F3H	—

这些参数在数据存储器（RAM）中的存放地址分别如下：

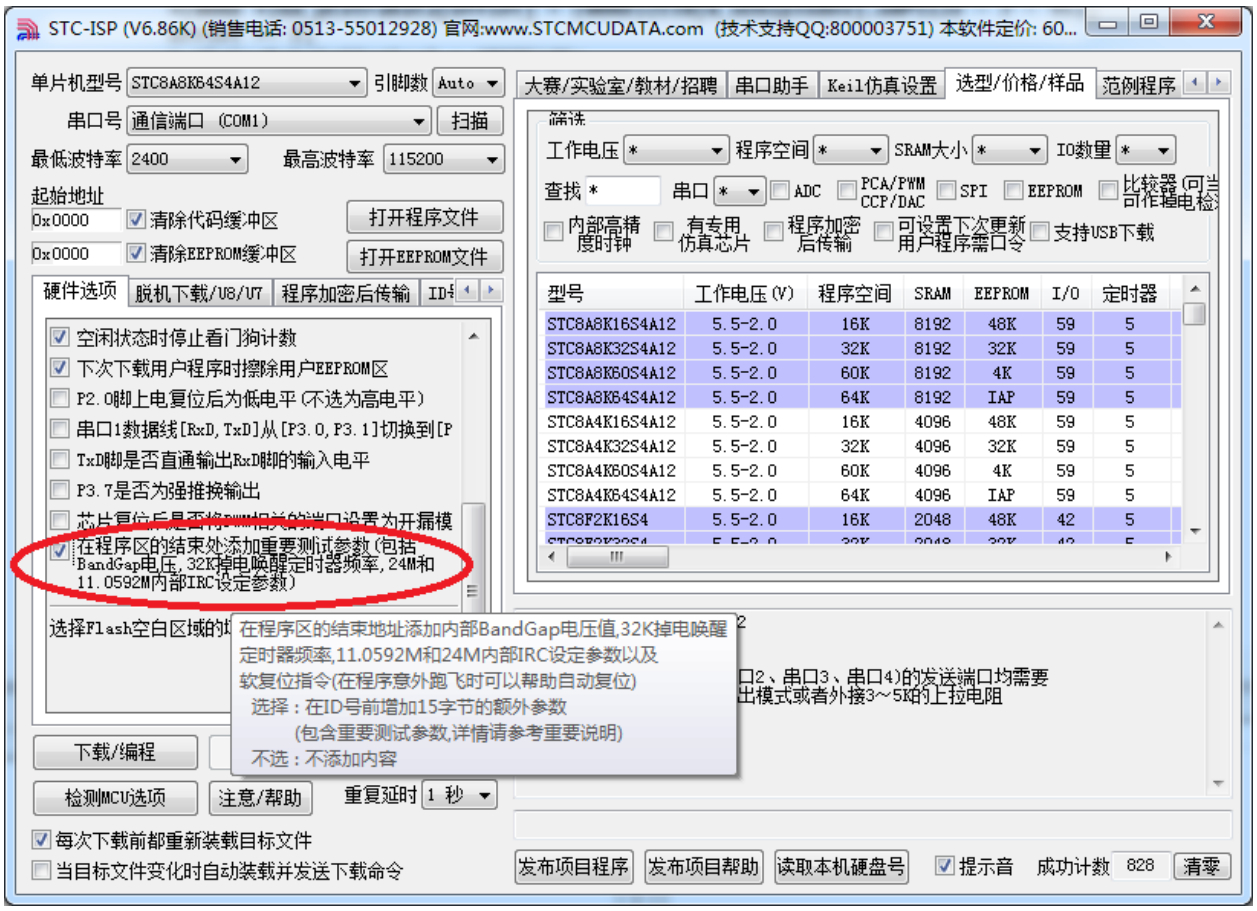
参数名称	保存地址	参数说明
内部 1.344V 参考信号源	idata：0EFH~0F0H	毫伏（高字节在前）
全球唯一 ID 号	idata：0F1H~0F7H	7 字节
32K 掉电唤醒定时器的频率	idata：0F8H~0F9H	Hz （高字节在前）
22.1184MHz 的 IRC 参数	idata：0FAH	—
24MHz 的 IRC 参数	idata：0FBH	—

特别说明

1、由于 RAM 中的参数可能被修改，所以一般不建议用户使用，特别是用户使用 ID 号进行加密时，强烈建议用于读取 ROM 中的 ID 数据。

2、由于 STC8A8K64S4A10、STC8A4K64S2A10、STC8F2K64S4 和 STC8F2K64S2 这 4 个型号的 EEPROM 的大小用户是可以自己设置的，有可能将保存重要参数的 ROM 空间设置为 EEPROM 而人为的将重要参数擦除或修改，所以使用这 4 个型号进行 ID 号进行加密时可能需要考虑这个问题。

3、默认情况下，程序存储器中只有全球唯一 ID 号的数据，而 Bandgap 电压值、32K 掉电唤醒定时器的频率以及 IRC 参数都是没有的，需要在 ISP 下载时选择如下图所示的选项才可用。



7.3.1 读取内部 1.344V参考信号源值 (从Flash 程序存储器（ROM）中读取)

汇编代码

```
AUXR    DATA    8EH
BGV     EQU     0FDF7H          ;STC8A8K64S4A10
```

```
;BGV        EQU     0EFF7H              ;STC8A8K60S4A10
;BGV        EQU     07FF7H              ;STC8A8K32S4A10
;BGV        EQU     03FF7H              ;STC8A8K16S4A10

BUSY        BIT     20H.0

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
UARTISR_EXIT:
            RETI

UART_INIT:
            MOV     SCON,#50H
            MOV     TMOD,#00H
            MOV     TL1,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     TH1,#0FFH
            SETB    TR1
            MOV     AUXR,#40H
            CLR     BUSY
            RET

UART_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     SBUF,A
            RET

MAIN:
            MOV     SP,#3FH

            LCALL   UART_INIT
            SETB    ES
            SETB    EA

            MOV     DPTR,#BGV
            CLR     A
            MOVC    A,@A+DPTR           ;读取 Bandgap 电压的高字节
            LCALL   UART_SEND
            MOV     A,#1
            MOVC    A,@A+DPTR           ;读取 Bandgap 电压的低字节
            LCALL   UART_SEND

LOOP:
            JMP     LOOP

            END
```

STC MCU

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

#define   FOSC              11059200UL
#define   BRT               (65536 - FOSC / 115200 / 4)

sfr       AUXR        =     0x8e;

bit       busy;
int       *BGV;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void main()
{
    BGV = (int code *)0xfdf7;           // STC8A8K64S4A10
//  BGV = (int code *)0xeff7;           // STC8A8K60S4A10
//  BGV = (int code *)0x7ff7;           // STC8A8K32S4A10
//  BGV = (int code *)0x3ff7;           // STC8A8K16S4A10
    UartInit();
    ES = 1;
    EA = 1;
    UartSend(*BGV >> 8);                //读取 Bandgap 电压的高字节
    UartSend(*BGV);                     //读取 Bandgap 电压的低字节

    while (1);
}
```

STC MCU

~~7.3.2 读取内部 1.344V参考信号源值 (从RAM中读取)~~

汇编代码

```
AUXR        DATA    8EH
BGV         DATA    0EFH

BUSY        BIT     20H.0

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
UARTISR_EXIT:
            RETI

UART_INIT:
            MOV     SCON,#50H
            MOV     TMOD,#00H
            MOV     TL1,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     TH1,#0FFH
            SETB    TR1
            MOV     AUXR,#40H
            CLR     BUSY
            RET

UART_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     SBUF,A
            RET

MAIN:
            MOV     SP,#3FH

            LCALL   UART_INIT
            SETB    ES
            SETB    EA

            MOV     R0,#BGV
            MOV     A,@R0               ;读取 Bandgap 电压的高字节
            LCALL   UART_SEND
            INC     R0
            MOV     A,@R0               ;读取 Bandgap 电压的低字节
            LCALL   UART_SEND

LOOP:
```

```
        JMP     LOOP

        END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

#define FOSC            11059200UL
#define BRT             (65536 - FOSC / 115200 / 4)

sfr     AUXR    =   0x8e;

bit     busy;
int     *BGV;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void main()
{
    BGV = (int idata *)0xef;
    UartInit();
    ES = 1;
    EA = 1;
    UartSend(*BGV >> 8);            //读取 Bandgap 电压的高字节
    UartSend(*BGV);                 //读取 Bandgap 电压的低字节

    while (1);
}
```

STC MCU

7.3.3 读取全球唯一ID号 (从Flash 程序存储器（ROM）中读取)

汇编代码

```
AUXR        DATA    8EH
ID          EQU     0FDF9H          ; STC8A8K64S4A10
;ID         EQU     0EFF9H          ; STC8A8K60S4A10
;ID         EQU     07FF9H          ; STC8A8K32S4A10
;ID         EQU     03FF9H          ; STC8A8K16S4A10

BUSY        BIT     20H.0

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
UARTISR_EXIT:
            RETI

UART_INIT:
            MOV     SCON,#50H
            MOV     TMOD,#00H
            MOV     TL1,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     TH1,#0FFH
            SETB    TR1
            MOV     AUXR,#40H
            CLR     BUSY
            RET

UART_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     SBUF,A
            RET

MAIN:
            MOV     SP,#3FH

            LCALL   UART_INIT
            SETB    ES
            SETB    EA

            MOV     DPTR,#ID
            MOV     R1,#7
NEXT:       CLR     A
```

```
                MOVC    A,@A+DPTR
                LCALL   UART_SEND
                INC     DPTR
                DJNZ    R1,NEXT

LOOP:
                JMP     LOOP

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

#define FOSC            11059200UL
#define BRT             (65536 - FOSC / 115200 / 4)

sfr AUXR        =   0x8e;

bit busy;
char *ID;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void main()
{
    char i;

    ID = (char code *)0xfdf9;           // STC8A8K64S4A10
```

```
//      ID = (char code *)0xeff9;               // STC8A8K60S4A10
//      ID = (char code *)0x7ff9;               // STC8A8K32S4A10
//      ID = (char code *)0x3ff9;               // STC8A8K16S4A10
        UartInit();
        ES = 1;
        EA = 1;

        for (i=0; i<7; i++)
        {
            UartSend(ID[i]);
        }

        while (1);
}
```

~~7.3.4 读取全球唯一ID号 (从RAM中读取)~~

汇编代码

```
AUXR        DATA    8EH
ID          DATA    0F1H

BUSY        BIT     20H.0

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
UARTISR_EXIT:
            RETI

UART_INIT:
            MOV     SCON,#50H
            MOV     TMOD,#00H
            MOV     TL1,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     TH1,#0FFH
            SETB    TR1
            MOV     AUXR,#40H
            CLR     BUSY
            RET

UART_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     SBUF,A
            RET
```

```
MAIN:
                MOV     SP,#3FH

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     R0,#ID
                MOV     R1,#7
NEXT:           MOV     A,@R0
                LCALL   UART_SEND
                INC     R0
                DJNZ    R1,NEXT

LOOP:
                JMP     LOOP

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

#define   FOSC              11059200UL
#define   BRT               (65536 - FOSC / 115200 / 4)

sfr       AUXR      =       0x8e;

bit       busy;
char      *ID;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
```

STC MCU

```
        while (busy);
        busy = 1;
        SBUF = dat;
}

void main()
{
        char i;

        ID = (char idata *)0xf1;
        UartInit();
        ES = 1;
        EA = 1;

        for (i=0; i<7; i++)
        {
                UartSend(ID[i]);
        }

        while (1);
}
```

7.3.5 读取 32K掉电唤醒定时器的频率 (从Flash 程序存储器（ROM）中读取)

汇编代码

```
AUXR        DATA    8EH
F32K        EQU     0FDF5H          ; STC8A8K64S4A10
;F32K       EQU     0EFF5H          ; STC8A8K60S4A10
;F32K       EQU     07FF5H          ; STC8A8K32S4A10
;F32K       EQU     03FF5H          ; STC8A8K16S4A10

BUSY        BIT     20H.0

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
UARTISR_EXIT:
            RETI

UART_INIT:
            MOV     SCON,#50H
```

```
                MOV     TMOD,#00H
                MOV     TL1,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     TH1,#0FFH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                RET

UART_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     SBUF,A
                RET

MAIN:
                MOV     SP,#3FH

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#F32K
                CLR     A
                MOVC    A,@A+DPTR               ;读取 32K 频率的高字节
                LCALL   UART_SEND
                INC     DPTR
                CLR     A
                MOVC    A,@A+DPTR               ;读取 32K 频率的低字节
                LCALL   UART_SEND

LOOP:
                JMP     LOOP

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

#define     FOSC            11059200UL
#define     BRT             (65536 - FOSC / 115200 / 4)

sfr         AUXR        =   0x8e;

bit         busy;
int         *F32K;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
```

```
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void main()
{
    F32K = (int code *)0xfdf5;          // STC8A8K64S4A10
//  F32K = (int code *)0xeff5;          // STC8A8K60S4A10
//  F32K = (int code *)0x7ff5;          // STC8A8K32S4A10
//  F32K = (int code *)0x3ff5;          // STC8A8K16S4A10
    UartInit();
    ES = 1;
    EA = 1;

    UartSend(*F32K >> 8);               //读取 32K 频率的高字节
    UartSend(*F32K);                    //读取 32K 频率的低字节

    while (1);
}
```

~~7.3.6 读取 32K掉电唤醒定时器的频率 (从RAM中读取)~~

汇编代码

```
AUXR        DATA    8EH
F32K        DATA    0F8H

BUSY        BIT     20H.0

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
```

```
CHKRI:
                JNB     RI,UARTISR_EXIT
                CLR     RI
UARTISR_EXIT:
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#00H
                MOV     TL1,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     TH1,#0FFH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                RET

UART_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     SBUF,A
                RET

MAIN:
                MOV     SP,#3FH

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     R0,#F32K
                MOV     A,@R0                   ;读取 32K 频率的高字节
                LCALL   UART_SEND
                INC     R0
                MOV     A,@R0                   ;读取 32K 频率的低字节
                LCALL   UART_SEND

LOOP:
                JMP     LOOP

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

#define     FOSC            11059200UL
#define     BRT             (65536 - FOSC / 115200 / 4)

sfr         AUXR        =   0x8e;

bit         busy;
int         *F32K;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
```

```
            busy = 0;
        }
        if (RI)
        {
            RI = 0;
        }
}

void UartInit()
{
        SCON = 0x50;
        TMOD = 0x00;
        TL1 = BRT;
        TH1 = BRT >> 8;
        TR1 = 1;
        AUXR = 0x40;
        busy = 0;
}

void UartSend(char dat)
{
        while (busy);
        busy = 1;
        SBUF = dat;
}

void main()
{
        F32K = (int idata *)0xf8;
        UartInit();
        ES = 1;
        EA = 1;

        UartSend(*F32K >> 8);           //读取 32K 频率的高字节
        UartSend(*F32K);                //读取 32K 频率的低字节

        while (1);
}
```

7.3.7 用户自定义内部IRC频率 (从Flash 程序存储器（ROM）中读取)

汇编代码

```
P_SW2       DATA    0BAH
CKSEL       EQU     0FE00H
CLKDIV      EQU     0FE01H

IRCCR       DATA    09FH

IRC22M      EQU     0FDF4H          ; STC8A8K64S4A10
IRC24M      EQU     0FDF3H
;IRC22M     EQU     0EFF4H          ; STC8A8K60S4A10
;IRC24M     EQU     0EFF3H
```

```
;IRC22M     EQU     07FF4H          ; STC8A8K32S4A10
;IRC24M     EQU     07FF3H
;IRC22M     EQU     03FF4H          ; STC8A8K16S4A10
;IRC24M     EQU     03FF3H

           ORG     0000H
           LJMP    MAIN

           ORG     0100H
MAIN:
           MOV     SP,#3FH

;          MOV     DPTR,#IRC22M    ;装载 22.1184MHz 的 IRC 参数
;          CLR     A
;          MOVC    A,@A+DPTR
;          MOV     IRCCR,A
           MOV     DPTR,#IRC24M    ;装载 24MHz 的 IRC 参数
           CLR     A
           MOVC    A,@A+DPTR
           MOV     IRCCR,A

           MOV     P_SW2,#80H
           MOV     A,#0            ;主时钟不预分频
           MOV     DPTR,#CLKDIV
           MOVX    @DPTR,A
           MOV     A,#40H          ;主时钟 4 分频输出到 P5.4 口
           MOV     DPTR,#CKSEL
           MOVX    @DPTR,A
           MOV     P_SW2,#00H

           JMP     $

           END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

#define    CKSEL           (*(unsigned char volatile xdata *)0xfe00)
#define    CLKDIV          (*(unsigned char volatile xdata *)0xfe01)

sfr        P_SW2       =   0xba;
sfr        IRCCR       =   0x9f;

char       *IRC22M;
char       *IRC24M;

void main()
{
    IRC22M = (char code *)0xfdf4;       // STC8A8K64S4A10
    IRC24M = (char code *) 0xfdf3;
//  IRC22M = (char code *)0xeff4;       // STC8A8K60S4A10
//  IRC24M = (char code *) 0xeff3;
//  IRC22M = (char code *)0x7ff4;       // STC8A8K32S4A10
//  IRC24M = (char code *) 0x7ff3;
//  IRC22M = (char code *)0x3ff4;       // STC8A8K16S4A10
//  IRC24M = (char code *) 0x3ff3;
```

```
//      IRCCR = *IRC22M;                //装载 22.1184MHz 的 IRC 参数
        IRCCR = *IRC24M;                //装载 24MHz 的 IRC 参数

        P_SW2 = 0x80;
        CLKDIV = 0;                     //主时钟不预分频
        CKSEL = 0x40;                   //主时钟 4 分频输出到 P5.4 口
        P_SW2 = 0x00;

        while (1);
}
```

~~7.3.8 用户自定义内部IRC频率 (从RAM中读取)~~

汇编代码

```
P_SW2           DATA    0BAH
CKSEL           EQU     0FE00H
CLKDIV          EQU     0FE01H

IRCCR           DATA    09FH

IRC22M          DATA    0FAH
IRC24M          DATA    0FBH

                ORG     0000H
                LJMP    MAIN

                ORG     0100H
MAIN:
                MOV     SP,#3FH

;               MOV     R0,#IRC22M          ;装载 22.1184MHz 的 IRC 参数
;               MOV     IRCCR,@R0
                MOV     R0,#IRC24M          ;装载 24MHz 的 IRC 参数
                MOV     IRCCR,@R0

                MOV     P_SW2,#80H
                MOV     A,#0                ;主时钟不预分频
                MOV     DPTR,#CLKDIV
                MOVX    @DPTR,A
                MOV     A,#40H              ;主时钟 4 分频输出到 P5.4 口
                MOV     DPTR,#CKSEL
                MOVX    @DPTR,A
                MOV     P_SW2,#00H

                JMP     $

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

#define     CKSEL           (*(unsigned char volatile xdata *)0xfe00)
#define     CLKDIV          (*(unsigned char volatile xdata *)0xfe01)
```

```
sfr     P_SW2       =   0xba;
sfr     IRCCR       =   0x9f;

char    *IRC22M;
char    *IRC24M;

void main()
{
    IRC22M = (char idata *)0xfa;
    IRC24M = (char idata *) 0xfb;
//  IRCCR = *IRC22M;            //装载 22.1184MHz 的 IRC 参数
    IRCCR = *IRC24M;            //装载 24MHz 的 IRC 参数

    P_SW2 = 0x80;
    CLKDIV = 0;                 //主时钟不预分频
    CKSEL = 0x40;               //主时钟 4 分频输出到 P5.4 口
    P_SW2 = 0x00;

    while (1);
}
```

STC MCU

8 特殊功能寄存器

8.1 STC8A8K64S4A12-64Pin系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H	P7	CH	CCAP0H	CCAP1H	CCAP2H	CCAP3H	PWMCR	RSTCFG
F0H	B	PWMCFG	PCA_PWM0	PCA_PWM1	PCA_PWM2	PCA_PWM3	PWMIF	PWMFDCR
E8H	P6	CL	CCAP0L	CCAP1L	CCAP2L	CCAP3L	~~IP3H~~	AUXINTIF
E0H	ACC	P7M1	P7M0	DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2	CCAPM3	ADCCFG	~~IP3~~
D0H	PSW	T4T3M	T4H	T4L	T3H	T3L	T2H	T2L
C8H	P5	P5M1	P5M0	P6M1	P6M0	SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2	VOCTRL	ADC_CONTR	ADC_RES	ADC_RESL	~~ADC_RESH~~
B0H	P3	P3M1	P3M0	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH	S3CON	S3BUF	TA	IE2
A0H	P2	BUS_SPEED	P_SW1	Reserved				Reserved
98H	SCON	SBUF	S2CON	S2BUF	Reserved		LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0	P0M1	P0M0	P2M1	P2M0	
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH	S4CON	S4BUF		PCON

可位寻址　　不可位寻址

注意：寄存器地址能够被 8 整除的才可进行位寻址，不能被 8 整除的则不可位寻址

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FFF0H	PWMCH	PWMCL	PWMCKS	TADCPH	TADCPL			
FF70H	PWM7T1H	PWM7T1L	PWM7T2H	PWM7T2L	PWM7CR	PWM7HLD		
FF60H	PWM6T1H	PWM6T1L	PWM6T2H	PWM6T2L	PWM6CR	PWM6HLD		
FF50H	PWM5T1H	PWM5T1L	PWM5T2H	PWM5T2L	PWM5CR	PWM5HLD		
FF40H	PWM4T1H	PWM4T1L	PWM4T2H	PWM4T2L	PWM4CR	PWM4HLD		
FF30H	PWM3T1H	PWM3T1L	PWM3T2H	PWM3T2L	PWM3CR	PWM3HLD		
FF20H	PWM2T1H	PWM2T1L	PWM2T2H	PWM2T2L	PWM2CR	PWM2HLD		
FF10H	PWM1T1H	PWM1T1L	PWM1T2H	PWM1T2L	PWM1CR	PWM1HLD		
FF00H	PWM0T1H	PWM0T1L	PWM0T2H	PWM0T2L	PWM0CR	PWM0HLD		
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
FE00H	CKSEL	CLKDIV	IRC24MCR	XOSCCR	IRC32KCR			

8.2 STC8A4K64S2A12-64Pin系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H	P7	CH	CCAP0H	CCAP1H	CCAP2H	CCAP3H	PWMCR	RSTCFG
F0H	B	PWMCFG	PCA_PWM0	PCA_PWM1	PCA_PWM2	PCA_PWM3	PWMIF	PWMFDCR
E8H	P6	CL	CCAP0L	CCAP1L	CCAP2L	CCAP3L	~~IP3H~~	AUXINTIF
E0H	ACC	P7M1	P7M0	DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2	CCAPM3	ADCCFG	~~IP3~~
D0H	PSW	T4T3M	T4H	T4L	T3H	T3L	T2H	T2L
C8H	P5	P5M1	P5M0	P6M1	P6M0	SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2	VOCTRL	ADC_CONTR	ADC_RES	ADC_RESL	~~ADC_RESH~~
B0H	P3	P3M1	P3M0	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH			TA	IE2
A0H	P2	BUS_SPEED	P_SW1	Reserved				Reserved
98H	SCON	SBUF	S2CON	S2BUF	Reserved		LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0	P0M1	P0M0	P2M1	P2M0	
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH				PCON

可位寻址　　不可位寻址

注意：寄存器地址能够被 8 整除的才可进行位寻址，不能被 8 整除的则不可位寻址

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FFF0H	PWMCH	PWMCL	PWMCKS	TADCPH	TADCPL			
FF70H	PWM7T1H	PWM7T1L	PWM7T2H	PWM7T2L	PWM7CR	PWM7HLD		
FF60H	PWM6T1H	PWM6T1L	PWM6T2H	PWM6T2L	PWM6CR	PWM6HLD		
FF50H	PWM5T1H	PWM5T1L	PWM5T2H	PWM5T2L	PWM5CR	PWM5HLD		
FF40H	PWM4T1H	PWM4T1L	PWM4T2H	PWM4T2L	PWM4CR	PWM4HLD		
FF30H	PWM3T1H	PWM3T1L	PWM3T2H	PWM3T2L	PWM3CR	PWM3HLD		
FF20H	PWM2T1H	PWM2T1L	PWM2T2H	PWM2T2L	PWM2CR	PWM2HLD		
FF10H	PWM1T1H	PWM1T1L	PWM1T2H	PWM1T2L	PWM1CR	PWM1HLD		
FF00H	PWM0T1H	PWM0T1L	PWM0T2H	PWM0T2L	PWM0CR	PWM0HLD		
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
FE00H	CKSEL	CLKDIV	IRC24MCR	XOSCCR	IRC32KCR			

8.3 STC8F2K64S4-44Pin系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H	P7							RSTCFG
F0H	B	PWMCFG						
E8H	P6						~~IP3H~~	AUXINTIF
E0H	ACC	P7M1	P7M0	DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H								~~IP3~~
D0H	PSW	T4T3M	T4H	T4L	T3H	T3L	T2H	T2L
C8H	P5	P5M1	P5M0	P6M1	P6M0	SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2	VOCTRL				
B0H	P3	P3M1	P3M0	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH	S3CON	S3BUF	TA	IE2
A0H	P2	BUS_SPEED	P_SW1	Reserved				Reserved
98H	SCON	SBUF	S2CON	S2BUF	Reserved		LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0	P0M1	P0M0	P2M1	P2M0	
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH	S4CON	S4BUF		PCON

可位寻址　　不可位寻址

注意：寄存器地址能够被 8 整除的才可进行位寻址，不能被 8 整除的则不可位寻址

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
FE00H	CKSEL	CLKDIV	IRC24MCR	XOSCCR	IRC32KCR			

8.4 STC8F2K64S2-44Pin系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H	P7							RSTCFG
F0H	B	PWMCFG						
E8H	P6						~~IP3H~~	AUXINTIF
E0H	ACC	P7M1	P7M0	DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H								~~IP3~~
D0H	PSW	T4T3M	T4H	T4L	T3H	T3L	T2H	T2L
C8H	P5	P5M1	P5M0	P6M1	P6M0	SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2	VOCTRL				
B0H	P3	P3M1	P3M0	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH			TA	IE2
A0H	P2	BUS_SPEED	P_SW1	Reserved				Reserved
98H	SCON	SBUF	S2CON	S2BUF	Reserved		LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0	P0M1	P0M0	P2M1	P2M0	
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH				PCON

可位寻址　　不可位寻址

注意：寄存器地址能够被 8 整除的才可进行位寻址，不能被 8 整除的则不可位寻址

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
FE00H	CKSEL	CLKDIV	IRC24MCR	XOSCCR	IRC32KCR			

8.5 STC8F1K08S2-20Pin/16Pin系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H								RSTCFG
F0H	B							
E8H								AUXINTIF
E0H	ACC			DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H								
D0H	PSW						T2H	T2L
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H		WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2	VOCTRL				
B0H	P3	P3M1	P3M0			IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH			TA	IE2
A0H			P_SW1	Reserved				Reserved
98H	SCON	SBUF	S2CON	S2BUF	Reserved		LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0	P0M1	P0M0			
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH				PCON

可位寻址　　不可位寻址

注意：寄存器地址能够被 8 整除的才可进行位寻址，不能被 8 整除的则不可位寻址

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
FE00H	CKSEL	CLKDIV	IRC24MCR	XOSCCR	IRC32KCR			

8.6 STC8F1K08S-8Pin系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H								RSTCFG
F0H	B							
E8H								AUXINTIF
E0H	ACC			DPS	DPL1	DPH1		
D8H								
D0H	PSW						T2H	T2L
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H		WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2	VOCTRL				
B0H	P3	P3M1	P3M0			IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH			TA	IE2
A0H			P_SW1	Reserved				Reserved
98H	SCON	SBUF			Reserved		LIRTRIM	IRTRIM
90H								
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H		SP	DPL	DPH				PCON

可位寻址　　不可位寻址

注意：寄存器地址能够被 8 整除的才可进行位寻址，不能被 8 整除的则不可位寻址

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
FE00H	CKSEL	CLKDIV	IRC24MCR	XOSCCR	IRC32KCR			

8.7 特殊功能寄存器列表

注意：寄存器地址能够被 8 整除的才可进行位寻址，不能被 8 整除的则不可位寻址。

STC8G 系列能进行位寻址的寄存器：P0（80H）、TCON（88H）、P1（90H）、SCON（98H）、P2（A0H）、IE（A8H）、P3（B0H）、IP（B8H）、P4（C0H）、P5（C8H）、PSW（D0H）、CCON（D8H）、ACC（E0H）、P6（E8H)、B（F0H）、P7（F8H）

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	P0 端口	80H									1111,1111
SP	堆栈指针	81H									0000,0111
DPL	数据指针（低字节）	82H									0000,0000
DPH	数据指针（高字节）	83H									0000,0000
S4CON	串口 4 控制寄存器	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	串口 4 数据寄存器	85H									0000,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	定时器模式寄存器	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	定时器 0 低 8 为寄存器	8AH									0000,0000
TL1	定时器 1 低 8 为寄存器	8BH									0000,0000
TH0	定时器 0 高 8 为寄存器	8CH									0000,0000
TH1	定时器 1 高 8 为寄存器	8DH									0000,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
INTCLKO	中断与时钟输出控制寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
P1	P1 端口	90H									1111,1111
P1M1	P1 口配置寄存器 1	91H									0000,0000
P1M0	P1 口配置寄存器 0	92H									0000,0000
P0M1	P0 口配置寄存器 1	93H									0000,0000
P0M0	P0 口配置寄存器 0	94H									0000,0000
P2M1	P2 口配置寄存器 1	95H									0000,0000
P2M0	P2 口配置寄存器 0	96H									0000,0000
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	串口 1 数据寄存器	99H									0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S2BUF	串口 2 数据寄存器	9BH									0000,0000
LIRTRIM	IRC 频率微调寄存器	9EH	-	-	-	-	-	-	LIRTRIM[1:0]		0000,00nn
IRTRIM	IRC 频率调整寄存器	9FH	IRTRIM[7:0]								nnnn,nnnn
P2	P2 端口	A0H									1111,1111
BUS_SPEED	总线速度控制寄存器	A1H	RW_S[1:0]						SPEED[1:0]		00xx,xx00
P_SW1	外设端口切换寄存器 1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-	nn00,000x
IE	中断允许寄存器	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
SADDR	串口 1 从机地址寄存器	A9H									0000,0000
WKTCL	掉电唤醒定时器低字节	AAH									1111,1111
WKTCH	掉电唤醒定时器高字节	ABH	WKTEN								0111,1111

S3CON	串口 3 控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	串口 3 数据寄存器	ADH									0000,0000
TA	DPTR 时序控制寄存器	AEH									0000,0000
IE2	中断允许寄存器 2	AFH	~~ECAN~~	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000,0000
P3	P3 端口	B0H									1111,1111
P3M1	P3 口配置寄存器 1	B1H									n000,0000
P3M0	P3 口配置寄存器 0	B2H									n000,0000
P4M1	P4 口配置寄存器 1	B3H									0000,0000
P4M0	P4 口配置寄存器 0	B4H									0000,0000
IP2	中断优先级控制寄存器 2	B5H	~~PCAN~~	PI2C	PCMP	PX4	PPWMFD	PPWM	PSPI	PS2	x000,0000
IP2H	高中断优先级控制寄存器 2	B6H	~~PCANH~~	PI2CH	PCMPH	PX4H	PPWMFDH	PPWMH	PSPIH	PS2H	x000,0000
IPH	高中断优先级控制寄存器	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	0000,0000
IP	中断优先级控制寄存器	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000,0000
SADEN	串口 1 从机地址屏蔽寄存器	B9H									0000,0000
P_SW2	外设端口切换寄存器 2	BAH	EAXFR	~~CAN_S~~	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S	0x00,0000
VOCTRL	电压控制寄存器	BBH	SCC	-	-	-	-	-	0	0	0xxx,xx00
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]				000x,0000
ADC_RES	ADC 转换结果高位寄存器	BDH									0000,0000
ADC_RESL	ADC 转换结果低位寄存器	BEH									0000,0000
P4	P4 端口	C0H	P4[7:0]								1111,1111
WDT_CONTR	看门狗控制寄存器	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]			0x00,0000
IAP_DATA	IAP 数据寄存器	C2H									1111,1111
IAP_ADDRH	IAP 高地址寄存器	C3H									0000,0000
IAP_ADDRL	IAP 低地址寄存器	C4H									0000,0000
IAP_CMD	IAP 命令寄存器	C5H	-	-	-	-	-	-	CMD[1:0]		xxxx,xx00
IAP_TRIG	IAP 触发寄存器	C6H									0000,0000
IAP_CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	IAP_WT[2:0]			0000,x000
P5	P5 端口	C8H	-	-							xx11,1111
P5M1	P5 口配置寄存器 1	C9H	-	-							xx11,1111
P5M0	P5 口配置寄存器 0	CAH	-	-							xx11,1111
P6M1	P6 口配置寄存器 1	CBH									0000,0000
P6M0	P6 口配置寄存器 0	CCH									0000,0000
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPCTL	SPI 控制寄存器	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]		0000,0100
SPDAT	SPI 数据寄存器	CFH									0000,0000
PSW	程序状态字寄存器	D0H	CY	AC	F0	RS1	RS0	OV	-	P	0000,00x0
T4T3M	定时器 4/3 控制寄存器	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000
T4H	定时器 4 高字节	D2H									0000,0000
T4L	定时器 4 低字节	D3H									0000,0000
T3H	定时器 3 高字节	D4H									0000,0000
T3L	定时器 3 低字节	D5H									0000,0000
T2H	定时器 2 高字节	D6H									0000,0000
T2L	定时器 2 低字节	D7H									0000,0000
CCON	PCA 控制寄存器	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0	00xx,0000

STC MCU

CMOD	PCA 模式寄存器	D9H	CIDL	-	-	-	CPS[2:0]			ECF	0xxx,0000
CCAPM0	PCA 模块 0 模式控制寄存器	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA 模块 1 模式控制寄存器	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA 模块 2 模式控制寄存器	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
CCAPM3	PCA 模块 3 模式控制寄存器	DDH	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3	x000,0000
~~IP3~~		~~DFH~~	~~-~~	~~-~~	~~-~~	~~-~~	~~-~~	~~-~~	~~PS4~~	~~PS3~~	~~xxxx,xx00~~
ADCCFG	ADC 配置寄存器	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000
ACC	累加器	E0H									0000,0000
P7M1	P7 口配置寄存器 1	E1H									0000,0000
P7M0	P7 口配置寄存器 0	E2H									0000,0000
DPS	DPTR 指针选择器	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
DPL1	第二组数据指针（低字节）	E4H									0000,0000
DPH1	第二组数据指针（高字节）	E5H									0000,0000
CMPCR1	比较器控制寄存器 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	比较器控制寄存器 2	E7H	INVCMPO	DISFLT	LCDTY[5:0]						0000,0000
P6	P6 端口	E8H									1111,1111
CL	PCA 计数器低字节	E9H									0000,0000
CCAP0L	PCA 模块 0 低字节	EAH									0000,0000
CCAP1L	PCA 模块 1 低字节	EBH									0000,0000
CCAP2L	PCA 模块 2 低字节	ECH									0000,0000
CCAP3L	PCA 模块 3 低字节	EDH									0000,0000
~~IP3H~~	~~高中断优先级控制寄存器 3~~	~~EEH~~	~~-~~	~~-~~	~~-~~	~~-~~	~~-~~	~~-~~	~~PS4H~~	~~PS3H~~	~~xxxx,xx00~~
AUXINTIF	扩展外部中断标志寄存器	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
B	B 寄存器	F0H									0000,0000
PWMCFG	增强型 PWM 配置寄存器	F1H	CBIF	ETADC	-	-	-	-	-	-	00xx,xxxx
PCA_PWM0	PCA0 的 PWM 模式寄存器	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L	0000,0000
PCA_PWM1	PCA1 的 PWM 模式寄存器	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L	0000,0000
PCA_PWM2	PCA2 的 PWM 模式寄存器	F4H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L	0000,0000
PCA_PWM3	PCA3 的 PWM 模式寄存器	F5H	EBS3[1:0]		XCCAP3H[1:0]		XCCAP3L[1:0]		EPC3H	EPC3L	0000,0000
PWMIF	增强型 PWM 中断标志寄存器	F6H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWMFDCR	PWM 异常检测控制寄存器	F7H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
P7	P7 端口	F8H									1111,1111
CH	PCA 计数器高字节	F9H									0000,0000
CCAP0H	PCA 模块 0 高字节	FAH									0000,0000
CCAP1H	PCA 模块 1 高字节	FBH									0000,0000
CCAP2H	PCA 模块 2 高字节	FCH									0000,0000
CCAP3H	PCA 模块 3 高字节	FDH									0000,0000
PWMCR	PWM 控制寄存器	FEH	ENPWM	ECBI	-	-	-	-	-	-	00xx,xxxx
RSTCFG	复位配置寄存器	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]		0000,0000

下列特殊功能寄存器为扩展 SFR，逻辑地址位于 XDATA 区域，访问前需要将 P_SW2（BAH）寄存器的最高位（EAXFR）置 1，然后使用 MOVX A,@DPTR 和 MOVX @DPTR,A 指令进行访问

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWMCH	PWM 计数器高字节	FFF0H	-								x000,0000

PWMCL	PWM 计数器低字节	FFF1H									0000,0000
PWMCKS	PWM 时钟选择	FFF2H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
TADCPH	触发 ADC 计数值高字节	FFF3H	-								x000,0000
TADCPL	触发 ADC 计数值低字节	FFF4H									0000,0000
PWM0T1H	PWM0T1 计数值高字节	FF00H	-								x000,0000
PWM0T1L	PWM0T1 计数值低节	FF01H									0000,0000
PWM0T2H	PWM0T2 数值高字节	FF02H	-								x000,0000
PWM0T2L	PWM0T2 数值低节	FF03H									0000,0000
PWM0CR	PWM0 控制寄存器	FF04H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI	00x0,0000
PWM0HLD	PWM0 电平保持控制寄存器	FF05H	-	-	-	-	-	-	HC0H	HC0L	xxxx,xx00
PWM1T1H	PWM1T1 计数值高字节	FF10H	-								x000,0000
PWM1T1L	PWM1T1 计数值低节	FF11H									0000,0000
PWM1T2H	PWM1T2 数值高字节	FF12H	-								x000,0000
PWM1T2L	PWM1T2 数值低节	FF13H									0000,0000
PWM1CR	PWM1 控制寄存器	FF14H	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI	00x0,0000
PWM1HLD	PWM1 电平保持控制寄存器	FF15H	-	-	-	-	-	-	HC1H	HC1L	xxxx,xx00
PWM2T1H	PWM2T1 计数值高字节	FF20H	-								x000,0000
PWM2T1L	PWM2T1 计数值低节	FF21H									0000,0000
PWM2T2H	PWM2T2 数值高字节	FF22H	-								x000,0000
PWM2T2L	PWM2T2 数值低节	FF23H									0000,0000
PWM2CR	PWM2 控制寄存器	FF24H	ENC2O	C2INI	-	C2_S[1:0]		EC2I	EC2T2SI	EC2T1SI	00x0,0000
PWM2HLD	PWM2 电平保持控制寄存器	FF25H	-	-	-	-	-	-	HC2H	HC2L	xxxx,xx00
PWM3T1H	PWM3T1 计数值高字节	FF30H	-								x000,0000
PWM3T1L	PWM3T1 计数值低节	FF31H									0000,0000
PWM3T2H	PWM3T2 数值高字节	FF32H	-								x000,0000
PWM3T2L	PWM3T2 数值低节	FF33H									0000,0000
PWM3CR	PWM3 控制寄存器	FF34H	ENC3O	C3INI	-	C3_S[1:0]		EC3I	EC3T2SI	EC3T1SI	00x0,0000
PWM3HLD	PWM3 电平保持控制寄存器	FF35H	-	-	-	-	-	-	HC3H	HC3L	xxxx,xx00
PWM4T1H	PWM4T1 计数值高字节	FF40H	-								x000,0000
PWM4T1L	PWM4T1 计数值低节	FF41H									0000,0000
PWM4T2H	PWM4T2 数值高字节	FF42H	-								x000,0000
PWM4T2L	PWM4T2 数值低节	FF43H									0000,0000
PWM4CR	PWM4 控制寄存器	FF44H	ENC4O	C4INI	-	C4_S[1:0]		EC4I	EC4T2SI	EC4T1SI	00x0,0000
PWM4HLD	PWM4 电平保持控制寄存器	FF45H	-	-	-	-	-	-	HC4H	HC4L	xxxx,xx00
PWM5T1H	PWM5T1 计数值高字节	FF50H	-								x000,0000
PWM5T1L	PWM5T1 计数值低节	FF51H									0000,0000
PWM5T2H	PWM5T2 数值高字节	FF52H	-								x000,0000
PWM5T2L	PWM5T2 数值低节	FF53H									0000,0000
PWM5CR	PWM5 控制寄存器	FF54H	ENC5O	C5INI	-	C5_S[1:0]		EC5I	EC5T2SI	EC5T1SI	00x0,0000
PWM5HLD	PWM5 电平保持控制寄存器	FF55H	-	-	-	-	-	-	HC5H	HC5L	xxxx,xx00
PWM6T1H	PWM6T1 计数值高字节	FF60H	-								x000,0000
PWM6T1L	PWM6T1 计数值低节	FF61H									0000,0000
PWM6T2H	PWM6T2 数值高字节	FF62H	-								x000,0000
PWM6T2L	PWM6T2 数值低节	FF63H									0000,0000

PWM6CR	PWM6 控制寄存器	FF64H	ENC6O	C6INI	-	C6_S[1:0]		EC6I	EC6T2SI	EC6T1SI	00x0,0000
PWM6HLD	PWM6 电平保持控制寄存器	FF65H	-	-	-	-	-	-	HC6H	HC6L	xxxx,xx00
PWM7T1H	PWM7T1 计数值高字节	FF70H	-								x000,0000
PWM7T1L	PWM7T1 计数值低节	FF71H									0000,0000
PWM7T2H	PWM7T2 数值高字节	FF72H	-								x000,0000
PWM7T2L	PWM7T2 数值低节	FF73H									0000,0000
PWM7CR	PWM7 控制寄存器	FF74H	ENC7O	C7INI	-	C7_S[1:0]		EC7I	EC7T2SI	EC7T1SI	00x0,0000
PWM7HLD	PWM7 电平保持控制寄存器	FF75H	-	-	-	-	-	-	HC7H	HC7L	xxxx,xx00
I2CCFG	I^2C 配置寄存器	FE80H	ENI2C	MSSL	MSSPEED[6:1]						0000,0000
I2CMSCR	I^2C 主机控制寄存器	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I^2C 主机状态寄存器	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I^2C 从机控制寄存器	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I^2C 从机状态寄存器	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I^2C 从机地址寄存器	FE85H	SLADR[6:0]							MA	0000,0000
I2CTXD	I^2C 数据发送寄存器	FE86H									0000,0000
I2CRXD	I^2C 数据接收寄存器	FE87H									0000,0000
I2CMSAUX	I^2C 主机辅助控制寄存器	FE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0
P0PU	P0 口上拉电阻控制寄存器	FE10H									0000,0000
P1PU	P1 口上拉电阻控制寄存器	FE11H									0000,0000
P2PU	P2 口上拉电阻控制寄存器	FE12H									0000,0000
P3PU	P3 口上拉电阻控制寄存器	FE13H									0000,0000
P4PU	P4 口上拉电阻控制寄存器	FE14H									0000,0000
P5PU	P5 口上拉电阻控制寄存器	FE15H									0000,0000
P6PU	P6 口上拉电阻控制寄存器	FE16H									0000,0000
P7PU	P7 口上拉电阻控制寄存器	FE17H									0000,0000
P0NCS	P0 口施密特触发控制寄存器	FE18H									0000,0000
P1NCS	P1 口施密特触发控制寄存器	FE19H									0000,0000
P2NCS	P2 口施密特触发控制寄存器	FE1AH									0000,0000
P3NCS	P3 口施密特触发控制寄存器	FE1BH									0000,0000
P4NCS	P4 口施密特触发控制寄存器	FE1CH									0000,0000
P5NCS	P5 口施密特触发控制寄存器	FE1DH									0000,0000
P6NCS	P6 口施密特触发控制寄存器	FE1EH									0000,0000
P7NCS	P7 口施密特触发控制寄存器	FE1FH									0000,0000
CKSEL	时钟选择寄存器	FE00H	MCLKODIV[3:0]				MCLKO_S	-	MCKSEL[1:0]		0000,0000
CLKDIV	时钟分频寄存器	FE01H									0000,0100
IRC24MCR	内部 24M 振荡器控制寄存器	FE02H	ENIRC24M	-	-	-	-	-	-	IRC24MST	1xxx,xxx0
XOSCCR	外部晶振控制寄存器	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST	00xx,xxx0
IRC32KCR	内部 32K 振荡器控制寄存器	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0

STC MCU

9 I/O口

所有的 I/O 口均有 4 种工作模式：准双向口/弱上拉（标准 8051 输出口模式）、推挽输出/强上拉、高阻输入（电流既不能流入也不能流出）、开漏输出。可使用软件对 I/O 口的工作模式进行容易配置。

9.1 I/O口相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	P0 端口	80H	P07	P06	P05	P04	P03	P02	P01	P00	1111,1111
P1	P1 端口	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111,1111
P2	P2 端口	A0H	P27	P26	P25	P24	P23	P22	P21	P20	1111,1111
P3	P3 端口	B0H	P37	P36	P35	P34	P33	P32	P31	P30	1111,1111
P4	P4 端口	C0H	P47	P46	P45	P44	P43	P42	P41	P40	1111,1111
P5	P5 端口	C8H	-	-	P55	P54	P53	P52	P51	P50	xx11,1111
P6	P6 端口	E8H	P67	P66	P65	P64	P63	P62	P61	P60	1111,1111
P7	P7 端口	F8H	P77	P76	P75	P74	P73	P72	P71	P70	1111,1111
P0M1	P0 口配置寄存器 1	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1	1111,1111
P0M0	P0 口配置寄存器 0	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0	0000,0000
P1M1	P1 口配置寄存器 1	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1	1111,1111
P1M0	P1 口配置寄存器 0	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0	0000,0000
P2M1	P2 口配置寄存器 1	95H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1	1111,1111
P2M0	P2 口配置寄存器 0	96H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0	0000,0000
P3M1	P3 口配置寄存器 1	B1H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1	1111,1100
P3M0	P3 口配置寄存器 0	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0	0000,0000
P4M1	P4 口配置寄存器 1	B3H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1	1111,1111
P4M0	P4 口配置寄存器 0	B4H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0	0000,0000
P5M1	P5 口配置寄存器 1	C9H	-	-	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1	xx11,1111
P5M0	P5 口配置寄存器 0	CAH	-	-	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0	xx00,0000
P6M1	P6 口配置寄存器 1	CBH	P67M1	P66M1	P65M1	P64M1	P63M1	P62M1	P61M1	P60M1	0000,0000
P6M0	P6 口配置寄存器 0	CCH	P67M0	P66M0	P65M0	P64M0	P63M0	P62M0	P61M0	P60M0	0000,0000
P7M1	P7 口配置寄存器 1	E1H	P77M1	P76M1	P75M1	P74M1	P73M1	P72M1	P71M1	P70M1	0000,0000
P7M0	P7 口配置寄存器 0	E2H	P77M0	P76M0	P75M0	P74M0	P73M0	P72M0	P71M0	P70M0	0000,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0PU	P0 口上拉电阻控制寄存器	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU	0000,0000
P1PU	P1 口上拉电阻控制寄存器	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU	0000,0000
P2PU	P2 口上拉电阻控制寄存器	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU	0000,0000
P3PU	P3 口上拉电阻控制寄存器	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU	0000,0000
P4PU	P4 口上拉电阻控制寄存器	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU	0000,0000
P5PU	P5 口上拉电阻控制寄存器	FE15H	-	-	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU	xx00,0000
P6PU	P6 口上拉电阻控制寄存器	FE16H	P67PU	P66PU	P65PU	P64PU	P63PU	P62PU	P61PU	P60PU	0000,0000

P7PU	P7 口上拉电阻控制寄存器	FE17H	P77PU	P76PU	P75PU	P74PU	P73PU	P72PU	P71PU	P70PU	0000,0000
P0NCS	P0 口施密特触发控制寄存器	FE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS	0000,0000
P1NCS	P1 口施密特触发控制寄存器	FE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS	0000,0000
P2NCS	P2 口施密特触发控制寄存器	FE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS	0000,0000
P3NCS	P3 口施密特触发控制寄存器	FE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS	0000,0000
P4NCS	P4 口施密特触发控制寄存器	FE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS	0000,0000
P5NCS	P5 口施密特触发控制寄存器	FE1DH	-	-	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS	xx00,0000
P6NCS	P6 口施密特触发控制寄存器	FE1EH	P67NCS	P66NCS	P65NCS	P64NCS	P63NCS	P62NCS	P61NCS	P60NCS	0000,0000
P7NCS	P7 口施密特触发控制寄存器	FE1FH	P77NCS	P76NCS	P75NCS	P74NCS	P73NCS	P72NCS	P71NCS	P70NCS	0000,0000

9.1.1 端口数据寄存器（Px）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0	80H	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
P1	90H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
P2	A0H	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
P3	B0H	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
P4	C0H	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
P5	C8H	-	-	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0
P6	E8H	P6.7	P6.6	P6.5	P6.4	P6.3	P6.2	P6.1	P6.0
P7	F8H	P7.7	P7.6	P7.5	P7.4	P7.3	P7.2	P7.1	P7.0

读写端口状态

写 0：输出低电平到端口缓冲区

写 1：输出高电平到端口缓冲区

读：直接读端口管脚上的电平

9.1.2 端口模式配置寄存器（PxM0，PxM1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0M0	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0
P0M1	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1
P1M0	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0
P1M1	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1
P2M0	96H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0
P2M1	95H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1
P3M0	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0
P3M1	B1H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1
P4M0	B4H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0
P4M1	B3H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1
P5M0	CAH	-	-	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0
P5M1	C9H	-	-	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1
P6M0	CCH	P67M0	P66M0	P65M0	P64M0	P63M0	P62M0	P61M0	P60M0
P6M1	CBH	P67M1	P66M1	P65M1	P64M1	P63M1	P62M1	P61M1	P60M1

P7M0	E2H	P77M0	P76M0	P75M0	P74M0	P73M0	P72M0	P71M0	P70M0
P7M1	E1H	P77M1	P76M1	P75M1	P74M1	P73M1	P72M1	P71M1	P70M1

配置端口的模式

PnM1.x	PnM0.x	Pn.x 口工作模式
0	0	准双向口
0	1	推挽输出
1	0	高阻输入
1	1	开漏输出

9.1.3 端口上拉电阻控制寄存器（PxPU）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU
P4PU	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	FE15H	-	-	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU
P6PU	FE16H	P67PU	P66PU	P65PU	P64PU	P63PU	P62PU	P61PU	P60PU
P7PU	FE17H	P77PU	P76PU	P75PU	P74PU	P73PU	P72PU	P71PU	P70PU

端口内部4.1K上拉电阻控制位（注：P3.0和P3.1口上的上拉电阻可能会略小一些）

0：禁止端口内部的 4.1K 上拉电阻

1：使能端口内部的 4.1K 上拉电阻

9.1.4 端口施密特触发控制寄存器（PxNCS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0NCS	FE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS
P1NCS	FE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS
P2NCS	FE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS
P3NCS	FE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS
P4NCS	FE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS
P5NCS	FE1DH	-	-	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS
P6NCS	FE1EH	P67NCS	P66NCS	P65NCS	P64NCS	P63NCS	P62NCS	P61NCS	P60NCS
P7NCS	FE1FH	P77NCS	P76NCS	P75NCS	P74NCS	P73NCS	P72NCS	P71NCS	P70NCS

端口施密特触发控制位

0：使能端口的施密特触发功能。（上电复位后默认使能施密特触发）

1：禁止端口的施密特触发功能。

VCC=5.0V	最小值	最大值	
普通 IO 输入高电平	2.2V	-	打开施密特触发
普通 IO 输入低电平	-	1.4V	
普通 IO 输入高电平	1.6V	-	关闭施密特触发

普通 IO 输入低电平	-	1.5V	
复位脚输入高电平	2.2V	-	
复位脚输入低电平	-	1.8V	

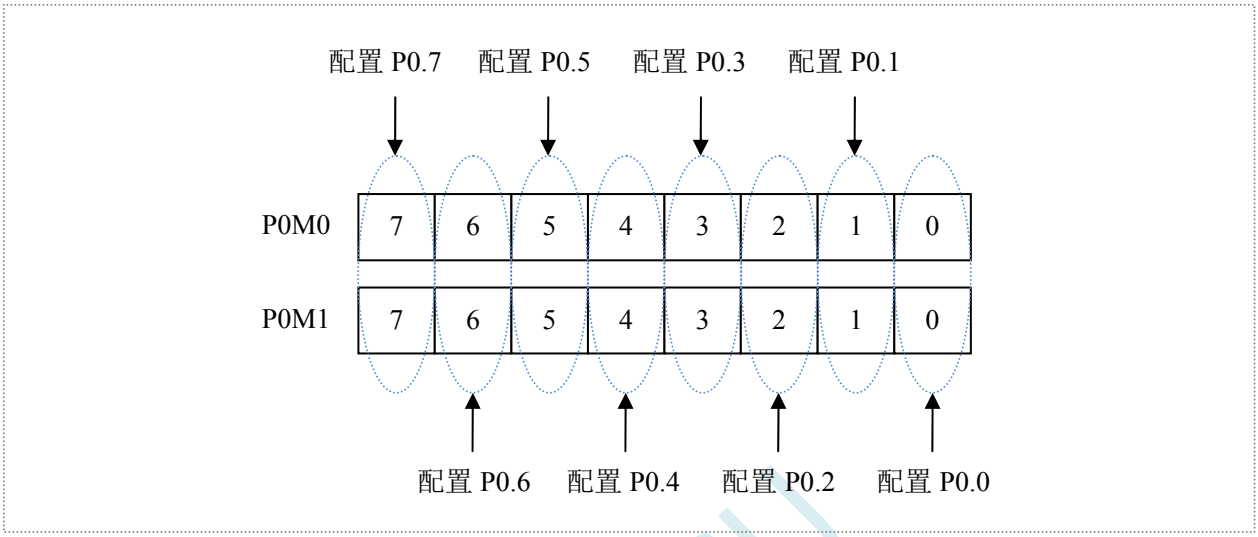
VCC=3.3V	最小值	最大值	
普通 IO 输入高电平	1.6V	-	打开施密特触发
普通 IO 输入低电平	-	1.0V	
普通 IO 输入高电平	1.2V		关闭施密特触发
普通 IO 输入低电平		1.1V	
复位脚输入高电平	1.7V	-	
复位脚输入低电平	-	1.3V	

STC MCU

9.2 配置I/O口

每个 I/O 的配置都需要使用两个寄存器进行设置。

以 P0 口为例，配置 P0 口需要使用 P0M0 和 P0M1 两个寄存器进行配置，如下图所示：



即 P0M0 的第 0 位和 P0M1 的第 0 位组合起来配置 P0.0 口的模式
即 P0M0 的第 1 位和 P0M1 的第 1 位组合起来配置 P0.1 口的模式
其他所有 I/O 的配置都与此类似。

PnM0 与 PnM1 的组合方式如下表所示

PnM1	PnM0	I/O 口工作模式
0	0	准双向口（传统8051端口模式，弱上拉） 灌电流可达20mA，拉电流为270～150μA（存在制造误差）
0	1	推挽输出（强上拉输出，可达20mA，要加限流电阻）
1	0	高阻输入（电流既不能流入也不能流出）
1	1	开漏输出（Open-Drain），内部上拉电阻断开 开漏模式既可读外部状态也可对外输出（高电平或低电平）。如要正确读外部状态或需要对外输出高电平，需外加上拉电阻，否则读不到外部状态，也对外输不出高电平。

注：n = 0, 1, 2, 3, 4, 5, 6, 7

注意：

虽然每个I/O口在弱上拉（准双向口）/强推挽输出/开漏模式时都能承受20mA的灌电流（还是要加限流电阻，如1K、560Ω、472Ω等），在强推挽输出时能输出20mA的拉电流（也要加限流电阻），但整个芯片的工作电流推荐不要超过70mA，即从Vcc流入的电流建议不要超过70mA，从Gnd流出电流建议不要超过70mA，整体流入/流出电流建议都不要超过70mA。（STC8G1K08A-8Pin系列和STC8G1K08-8Pin系列，整个芯片的工作电流推荐不要超过35mA，即从Vcc流入的电流建议不要超过35mA，从Gnd流出电流建议不要超过35mA，整体流入/流出电流建议都不要超过35mA）

STC MCU

9.3 I/O的结构图

9.3.1 准双向口（弱上拉）

准双向口（弱上拉）输出类型可用作输出和输入功能而不需重新配置端口输出状态。这是因为当端口输出为 1 时驱动能力很弱，允许外部装置将其拉低。当引脚输出为低时，它的驱动能力很强，可吸收相当大的电流。准双向口有 3 个上拉晶体管适应不同的需要。

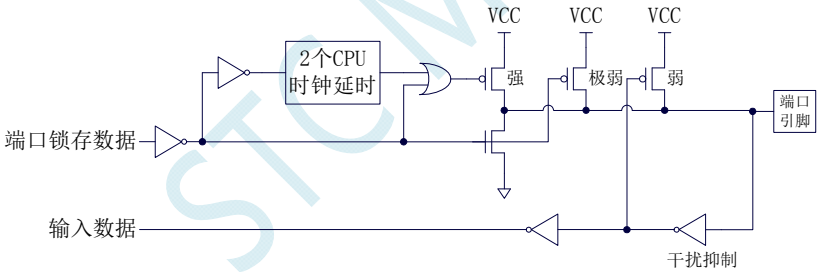
在 3 个上拉晶体管中，有 1 个上拉晶体管称为“弱上拉”，当端口寄存器为 1 且引脚本身也为 1 时打开。此上拉提供基本驱动电流使准双向口输出为 1。如果一个引脚输出为 1 而由外部装置下拉到低时，弱上拉关闭而“极弱上拉”维持开状态，为了把这个引脚强拉为低，外部装置必须有足够的灌电流能力使引脚上的电压降到门槛电压以下。对于 5V 单片机，“弱上拉”晶体管的电流约 250uA；对于 3.3V 单片机，“弱上拉”晶体管的电流约 150uA。

第 2 个上拉晶体管，称为“极弱上拉”，当端口锁存为 1 时打开。当引脚悬空时，这个极弱的上拉源产生很弱的上拉电流将引脚上拉为高电平。对于 5V 单片机，“极弱上拉”晶体管的电流约 18uA；对于 3.3V 单片机，“极弱上拉”晶体管的电流约 5uA。

第 3 个上拉晶体管称为“强上拉”。当端口锁存器由 0 到 1 跳变时，这个上拉用来加快准双向口由逻辑 0 到逻辑 1 转换。当发生这种情况时，强上拉打开约 2 个时钟以使引脚能够迅速地上拉到高电平。

准双向口（弱上拉）带有一个施密特触发输入以及一个干扰抑制电路。准双向口（弱上拉）读外部状态前,要先锁存为 ‘1’,才可读到外部正确的状态.

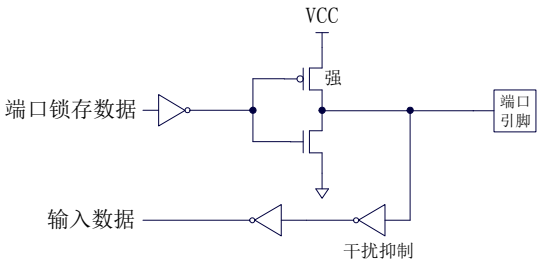
准双向口（弱上拉）输出如下图所示：



9.3.2 推挽输出

强推挽输出配置的下拉结构与开漏输出以及准双向口的下拉结构相同，但当锁存器为 1 时提供持续的强上拉。推挽模式一般用于需要更大驱动电流的情况。

强推挽引脚配置如下图所示：

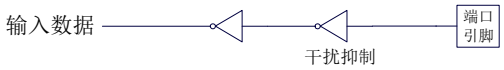


9.3.3 高阻输入

电流既不能流入也不能流出

输入口带有一个施密特触发输入以及一个干扰抑制电路

高阻输入引脚配置如下图所示：



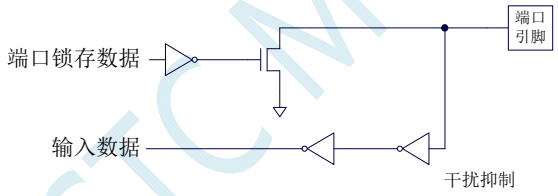
9.3.4 开漏输出

开漏模式既可读外部状态也可对外输出（高电平或低电平）。如要正确读外部状态或需要对外输出高电平，需外加上拉电阻。

当端口锁存器为 0 时，开漏输出关闭所有上拉晶体管。当作为一个逻辑输出高电平时，这种配置方式必须有外部上拉，一般通过电阻外接到 Vcc。如果外部有上拉电阻，开漏的 I/O 口还可读外部状态，即此时被配置为开漏模式的 I/O 口还可作为输入 I/O 口。这种方式的下拉与准双向口相同。

开漏端口带有一个施密特触发输入以及一个干扰抑制电路。

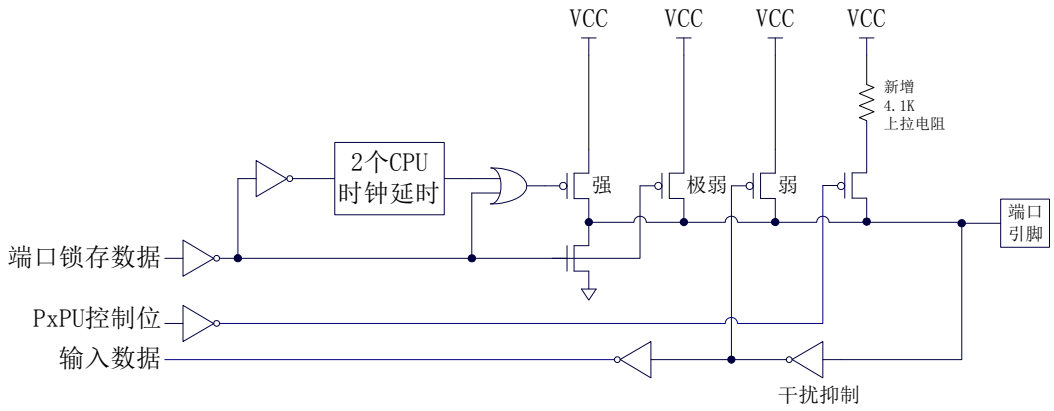
输出端口配置如下图所示：



STCMCU

9.3.5 新增 4.1K上拉电阻

STC8A/8F 系列所有的 I/O 口内部均可使能一个大约 4.1K 的上拉电阻（由于制造误差，上拉电阻的范围可能为 3K～5K）



端口上拉电阻控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU
P4PU	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	FE15H	-	-	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU
P6PU	FE16H	P67PU	P66PU	P65PU	P64PU	P63PU	P62PU	P61PU	P60PU
P7PU	FE17H	P77PU	P76PU	P75PU	P74PU	P73PU	P72PU	P71PU	P70PU

端口内部4.1K上拉电阻控制位（注：P3.0和P3.1口上的上拉电阻可能会略小一些）

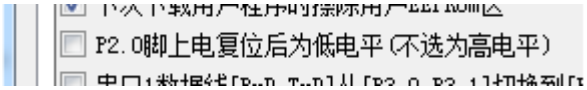
0：禁止端口内部的 4.1K 上拉电阻

1：使能端口内部的 4.1K 上拉电阻

9.4 特殊I/O口说明

9.4.1 P2.0/RSTCV

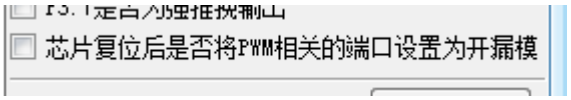
STC8A/8F 系列 P2.0 上电后初始电平可以使用 ISP 下载软件中的如下硬件选项进行设置



注意：当 MCU 的工作电压低于 1.6V 时，P2.0 输出电平高电平，只有当 MCU 的工作电压上升到 1.6V 以上时，P2.0 才会输出用户硬件选项设置的电平。

9.4.2 PWM相关I/O口

STC8A/8F 系列的所有 IO 口上电复位后默认模式均是弱上拉准双向口模式，用户可以通过 ISP 下载软件中的如下硬件选项将 PWM 相关的 IO 口配置为开漏模式



STC8A/8F 系列 PWM 相关的 I/O 口为 P1.0~P1.7，P2.0~P2.7，P6.0~P6.7

STC MCU

9.5 范例程序

9.5.1 端口模式设置

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M0    =   0x94;
sfr     P0M1    =   0x93;
sfr     P1M0    =   0x92;
sfr     P1M1    =   0x91;
sfr     P2M0    =   0x96;
sfr     P2M1    =   0x95;
sfr     P3M0    =   0xb2;
sfr     P3M1    =   0xb1;
sfr     P4M0    =   0xb4;
sfr     P4M1    =   0xb3;
sfr     P5M0    =   0xca;
sfr     P5M1    =   0xc9;
sfr     P6M0    =   0xcc;
sfr     P6M1    =   0xcb;
sfr     P7M0    =   0xe2;
sfr     P7M1    =   0xe1;

void main()
{
    P0M0 = 0x00;                    //设置 P0.0~P0.7 为双向口模式
    P0M1 = 0x00;
    P1M0 = 0xff;                    //设置 P1.0~P1.7 为推挽输出模式
    P1M1 = 0x00;
    P2M0 = 0x00;                    //设置 P2.0~P2.7 为高阻输入模式
    P2M1 = 0xff;
    P3M0 = 0xff;                    //设置 P3.0~P3.7 为开漏模式
    P3M1 = 0xff;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M0        DATA        094H
P0M1        DATA        093H
P1M0        DATA        092H
P1M1        DATA        091H
P2M0        DATA        096H
P2M1        DATA        095H
P3M0        DATA        0B2H
P3M1        DATA        0B1H
P4M0        DATA        0B4H
P4M1        DATA        0B3H
```

```
P5M0        DATA    0CAH
P5M1        DATA    0C9H
P6M0        DATA    0CCH
P6M1        DATA    0CBH
P7M0        DATA    0E2H
P7M1        DATA    0E1H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH

            MOV     P0M0,#00H           ;设置 P0.0~P0.7 为双向口模式
            MOV     P0M1,#00H
            MOV     P1M0,#0FFH          ;设置 P1.0~P1.7 为推挽输出模式
            MOV     P1M1,#00H
            MOV     P2M0,#00H           ;设置 P2.0~P2.7 为高阻输入模式
            MOV     P2M1,#0FFH
            MOV     P3M0,#0FFH          ;设置 P3.0~P3.7 为开漏模式
            MOV     P3M1,#0FFH

            JMP     $

            END
```

9.5.2 双向口读写操作

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M0        =   0x94;
sfr     P0M1        =   0x93;
sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;
sbit    P00         =   P0^0;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
```

```
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P0M0 = 0x00;                    //设置 P0.0~P0.7 为双向口模式
    P0M1 = 0x00;

    P00 = 1;                        //P0.0 口输出高电平
    P00 = 0;                        //P0.0 口输出低电平

    P00 = 1;                        //读取端口前先使能内部弱上拉电阻
    _nop_();                        //等待两个时钟
    _nop_();                        //
    CY = P00;                       //读取端口状态

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M0        DATA    094H
P0M1        DATA    093H
P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
```

```
MOV     P4M1, #00H
MOV     P5M0, #00H
MOV     P5M1, #00H

MOV     P0M0,#00H           ;设置 P0.0~P0.7 为双向口模式
MOV     P0M1,#00H

SETB    P0.0                ;P0.0 口输出高电平
CLR     P0.0                ;P0.0 口输出低电平

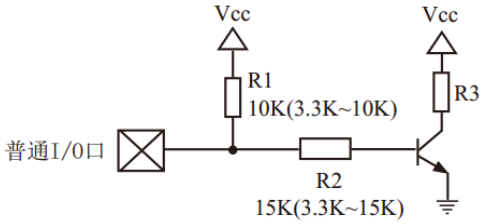
SETB    P0.0                ;读取端口前先使能内部弱上拉电阻
NOP                         ;等待两个时钟
NOP
MOV     C,P0.0              ;读取端口状态

JMP     $

END
```

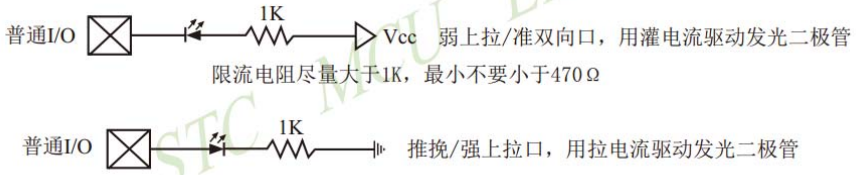
STC MCU

9.6 一种典型三极管控制电路



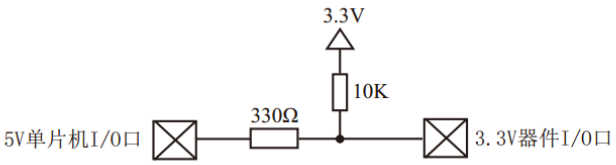
如果上拉控制，建议加上拉电阻 R1(3.3K～10K)，如果不加上拉电阻 R1(3.3K～10K)，建议 R2 的值在 15K 以上，或用强推挽输出。

9.7 典型发光二极管控制电路



9.8 混合电压供电系统 3V/5V器件I/O口互连

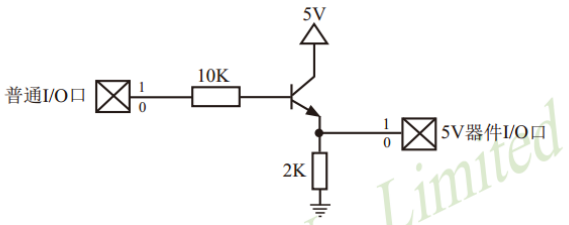
STC 的 5V 单片机连接 3.3V 器件时，为防止 3.3V 器件承受不了 5V，可将相应的 5V 单片机 I/O 口先串一个 330Ω 的限流电阻到 3.3V 器件 I/O 口，程序初始化时将 5V 单片机的 I/O 口设置成开漏配置，断开内部上拉电阻，相应的 3.3V 器件 I/O 口外部加 10K 上拉电阻到 3.3V 器件的 Vcc，这样高电平是 3.3V，低电平是 0V，输入输出一切正常。



STC 的 3V 单片机连接 5V 器件时，为防止 3V 单片机承受不了 5V，如果相应的 I/O 口是输入，可在该 I/O 口上串接一个隔离二极管，隔离高压部分。外部信号电压高于单片机工作电压时截止，I/O 口因内内部上拉到高电平，所以读 I/O 口状态是高电平；外部信号电压为低时导通，I/O 口被钳位在 0.7V，小于 0.8V 时单片机读 I/O 口状态是低电平。



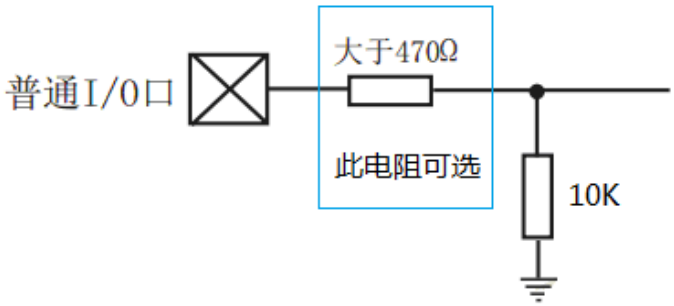
STC 的 3V 单片机连接 5V 器件时，为防止 3V 单片机承受不了 5V，如果相应的 I/O 口是输出，可用一个 NPN 三极管隔离，电路如下：



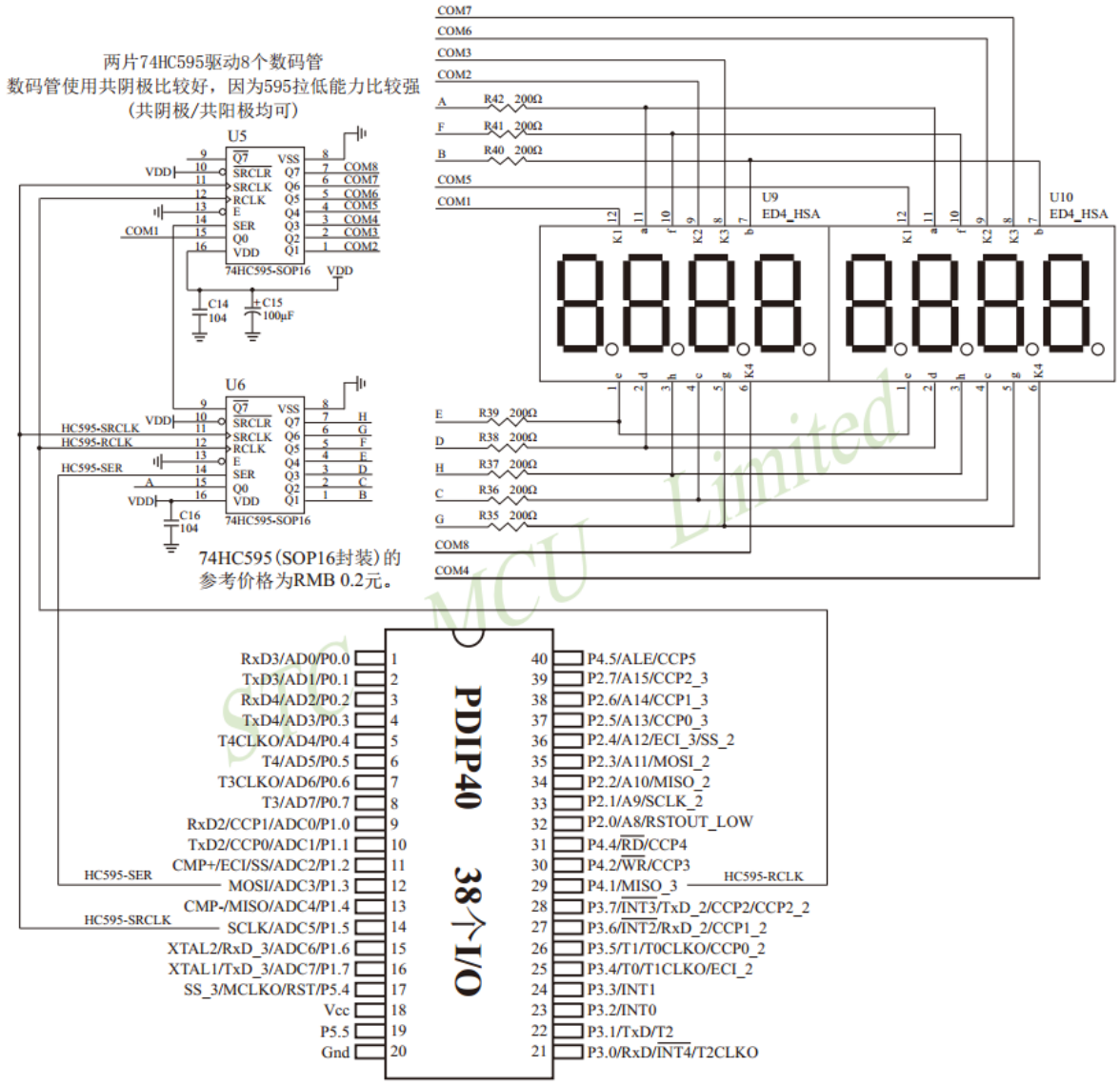
9.9 如何让I/O口上电复位时为低电平

传统 8051 单片机上电复位时普通 I/O 口为弱上拉(准双向口)高电平输出，而很多实际应用要求上电时某些 I/O 口为低电平输出，否则所控制的系统(如马达)就会误动作，新一代 STC8G 系列和 STC8H 系列单片机由于所有的 I/O 复位后是高阻输入(除 P3.0/P3.1 是传统的弱上拉)，加一个下拉电阻就可保证上电时为低电平，后续要改为高电平，只需要将 I/O 的模式改为强推挽输出，对外输出高电平即可。

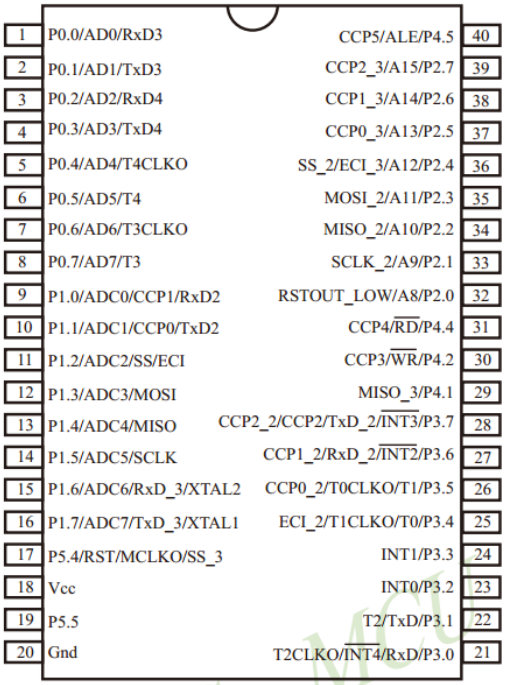
如下：在 STC 的单片机 I/O 口上加一个下拉电阻(10K 左右)，这样上电复位时，除了下载口 P3.0 和 P3.1 为弱上拉(准双向口)外，其他 I/O 口均为高阻输入模式，而外部有下拉电阻，所以该 I/O 口上电复位时外部为低电平。如果要将此 I/O 口驱动为高电平，可将此 I/O 口设置为强推挽输出，而强推挽输出时，I/O 口驱动电流可达 20mA，故肯定可以将该口驱动为高电平输出。

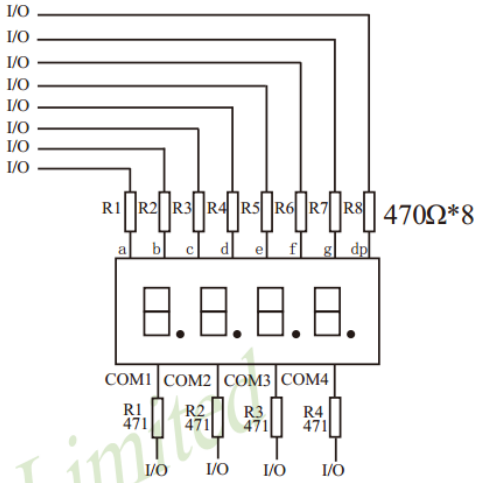


9.10 利用 74HC595 驱动 8 个数码管(串行扩展,3 根线)的线路图



9.11 I/O口直接驱动LED数码管应用线路图

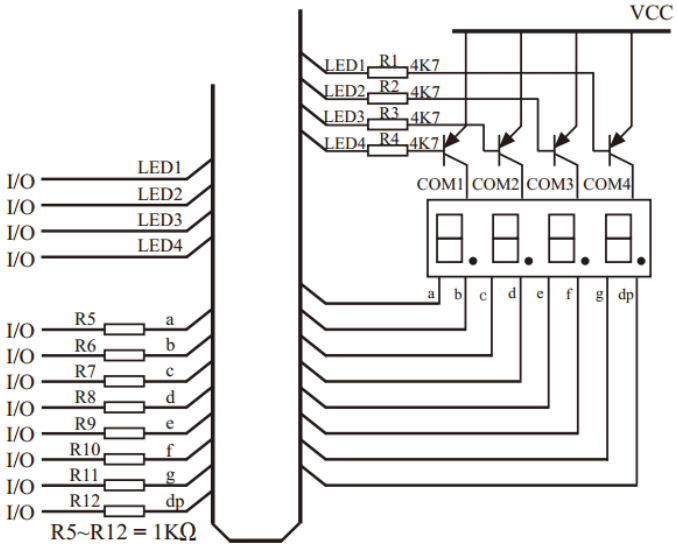




I/O口动态扫描驱动4个共阴极数码管参考电路图

I/O 口动态扫描驱动数码时，可以一次点亮一个数码管中的8段，但为降低功耗，建议可以一次只点亮其中的4段或者2段

I/O 口动态扫描驱动4个共阳极数码管参考电路图



9.12 用STC系列MCU的I/O口直接驱动段码LCD

当产品需要段码 LCD 显示时，如果使用不带 LCD 驱动器的 MCU，则需要外接 LCD 驱动 IC，这会增加成本。事实上，很多小项目，比如大量的小家电，需要显示的段码不多，常见的是 4 个 8 带小数点或时钟的冒号“:”，这样如果使用 IO 口直接扫描显示，则会降低成本，工作更可靠。

但是，本方案不合适驱动太多的段（ 占用 IO 太多），也不合适非常低功耗的场合（驱动会有几百 uA 电流）。

段码 LCD 驱动简单原理：如图 1 所示。

LCD 是一种特殊的液态晶体，在电场的作用下晶体的排列方向会发生扭转，因而改变其透光性，从而可以看到显示内容。LCD 有一个扭转电压阀值，当 LCD 两端电压高于此阀值时，显示内容，低于此阀值时，不显示。通常 LCD 有 3 个参数：工作电压、DUTY（对应 COM 数）和 BIAS（即偏压，对应阀值），比如 3.0V、1/4 DUTY、1/3 BIAS，表示 LCD 显示电压为 3.0V，4 个 COM，阀值大约是 1.5V，当加在某段 LCD 两端电压为 3.0V 时显示，而加 1.0V 时不显示。但是 LCD 对于驱动电压的反应不是很敏感的，比如加 2V 时，可能会微弱显示，这就是通常说的“鬼影”。所以要保证驱动显示时，要大于阀值电压比较多，而不显示时，要用比阀值小比较多的电压。

注意：LCD 要用交流驱动，其两端不能加直流电压，否则时间稍长就会损坏，所以要保证加在 LCD 两端的驱动电压的平均电压为 0。LCD 使用时分割扫描法，任何时候一个 COM 扫描有效，另外的 COM 处于无效状态。

驱动 1/4Duty 1/2BIAS 3V 的方案电路见图 1，LCD 扫描原理见图 3，MCU 为 3.0V 或 3.3V 工作，并且每个 COM 都串一个 20K 电阻接到一个电容 C1，RC 滤波后得到一个中点电压 1/2VDD。在轮到某个 COM 扫描时，连接的 IO 设置成推挽输出，其余 COM 设置成高阻，如果与本 COM 连接的 SEG 不显示，则 SEG 输出与 COM 同相，如果显示，则反相。扫描完后，这个 COM 的 IO 就设置成高阻。每个 COM 通过 20K 电阻连接到电容 C1 上的 1/2VDD 电压，而 SEG 根据是否显示输出高低电平，这样加在 LCD 段上的电压，显示时是+-VDD，不显示时是+-1/2VDD，保证了 LCD 两端平均直流电压为 0。

驱动 1/4Duty 1/3BIAS 3V 的方案电路见图 4，LCD 扫描原理见图 5，MCU 为 5V 工作，SEG 接的 IO 通过电阻分压输出 1.5V、3.5V，COM 接的 IO 通过电阻分压输出 0.5V、2.5V（高阻时）、4.5V，分压电阻公共点接到一个电容 C1，RC 滤波后得到一个中点电压 1/2VDD。在轮到某个 COM 扫描时，设置成推挽输出，如果与本 COM 连接的 SEG 不显示，则 SEG 输出与 COM 同相，如果显示，则反相。扫描完后，这个 COM 的 IO 就设置成高阻，这样这个 COM 就通过 47K 电阻连接到 2.5V 电压，而 SEG 根据是否显示输出高低电平，这样加在 LCD 上的电压，显示时是+-3.0V，不显示时是+-1.0V，完全满足 LCD 的扫描要求。

当需要睡眠省电时，把所有 COM 和 SEG 驱动 IO 全部输出低电平，LCD 驱动部分不会增加额外电流。

图 1：驱动 1/4Duty 1/2BIAS 3V LCD 的电路

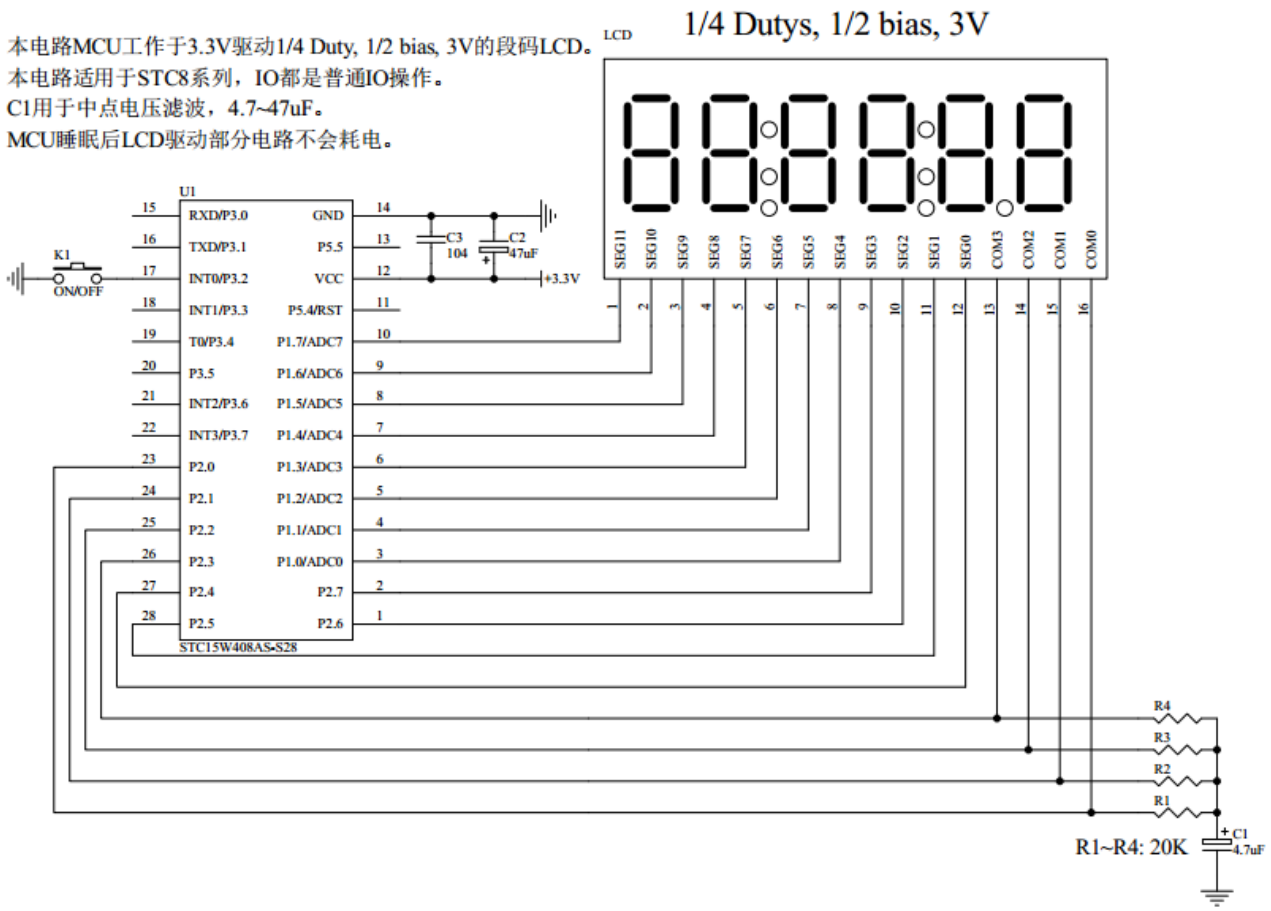


图 2：段码名称图

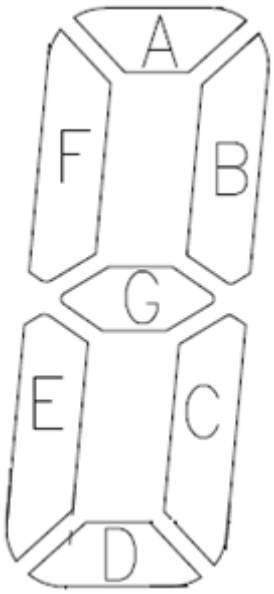


图 3：1/4Duty 1/2BIAS 扫描原理图

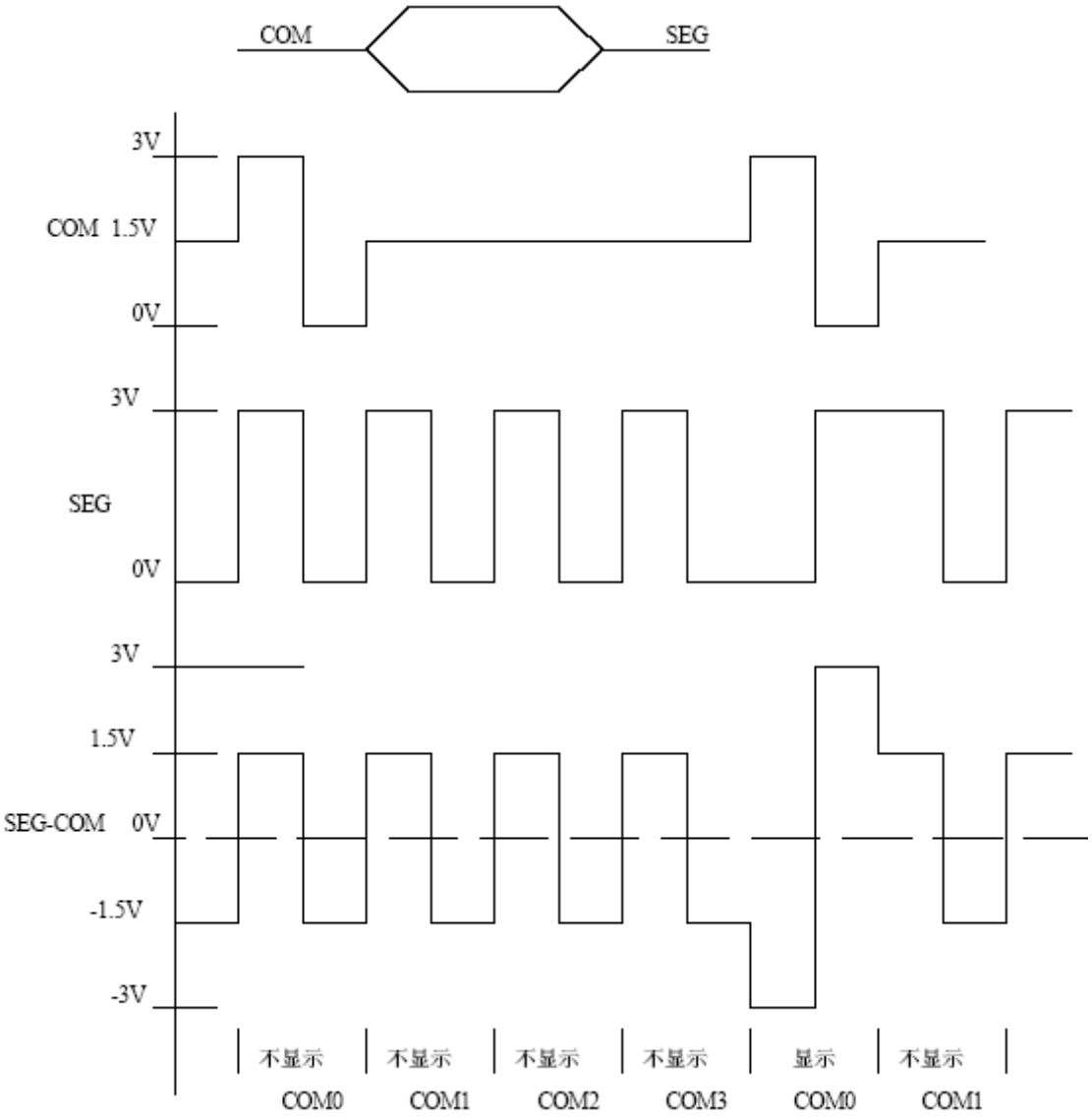


图 4：驱动 1/4Duty 1/3BIAS 3V LCD 的电路

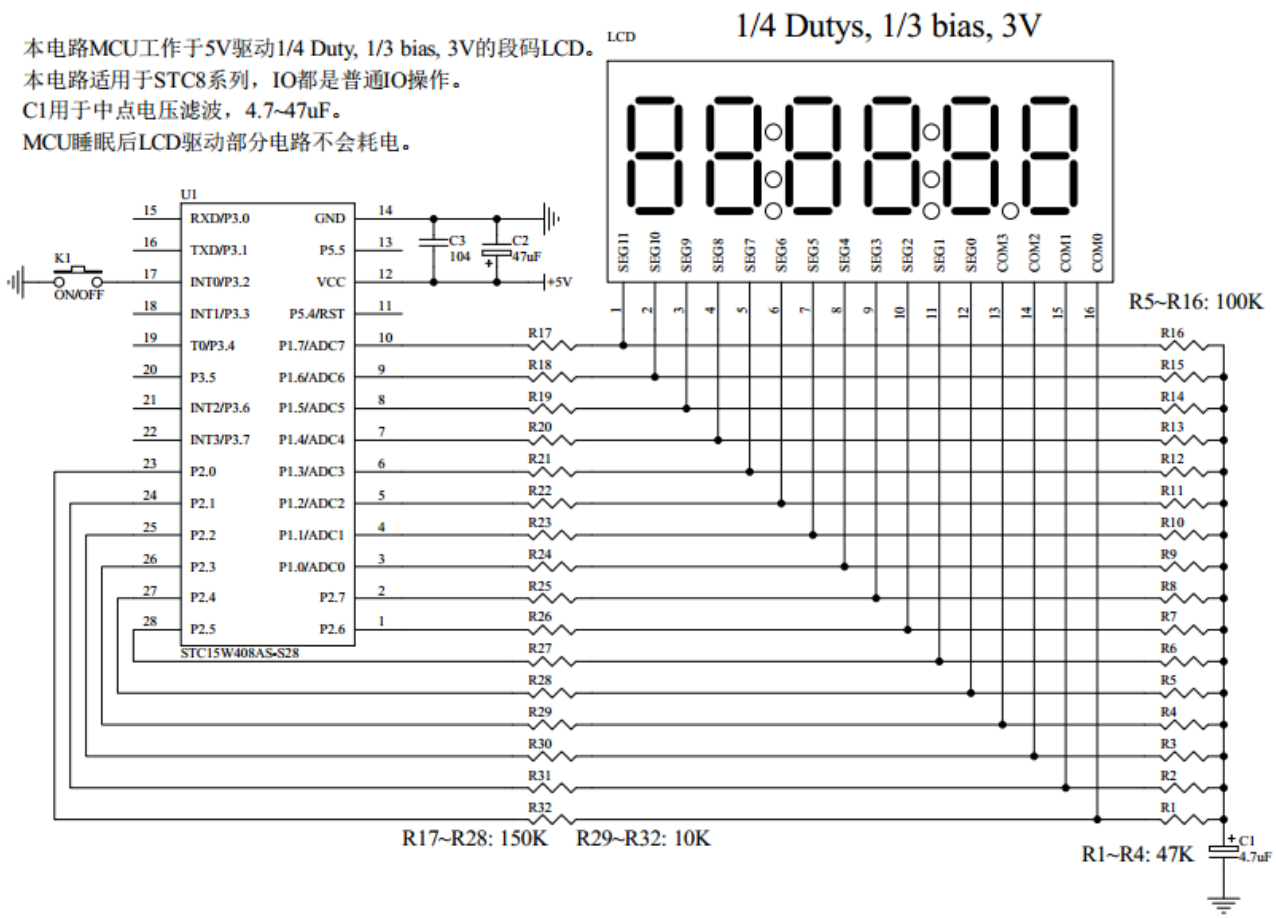
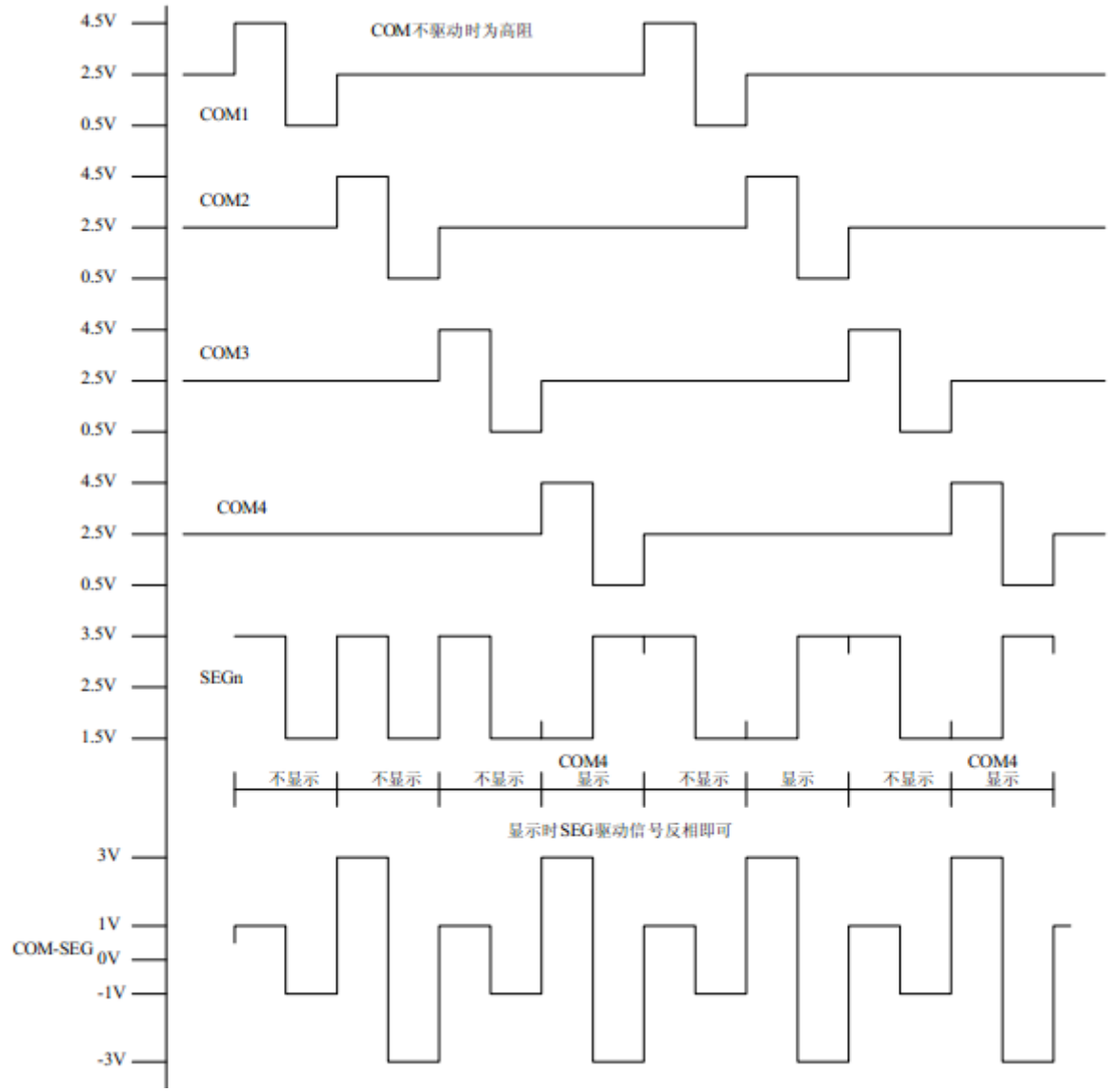


图 5：1/4Duty 1/3BIAS 扫描原理图



为了使用方便，显示内容放在一个显存中，其中的各个位与 LCD 的段一一对应，见图 6。

图 6：LCD 真值表和显存影射表

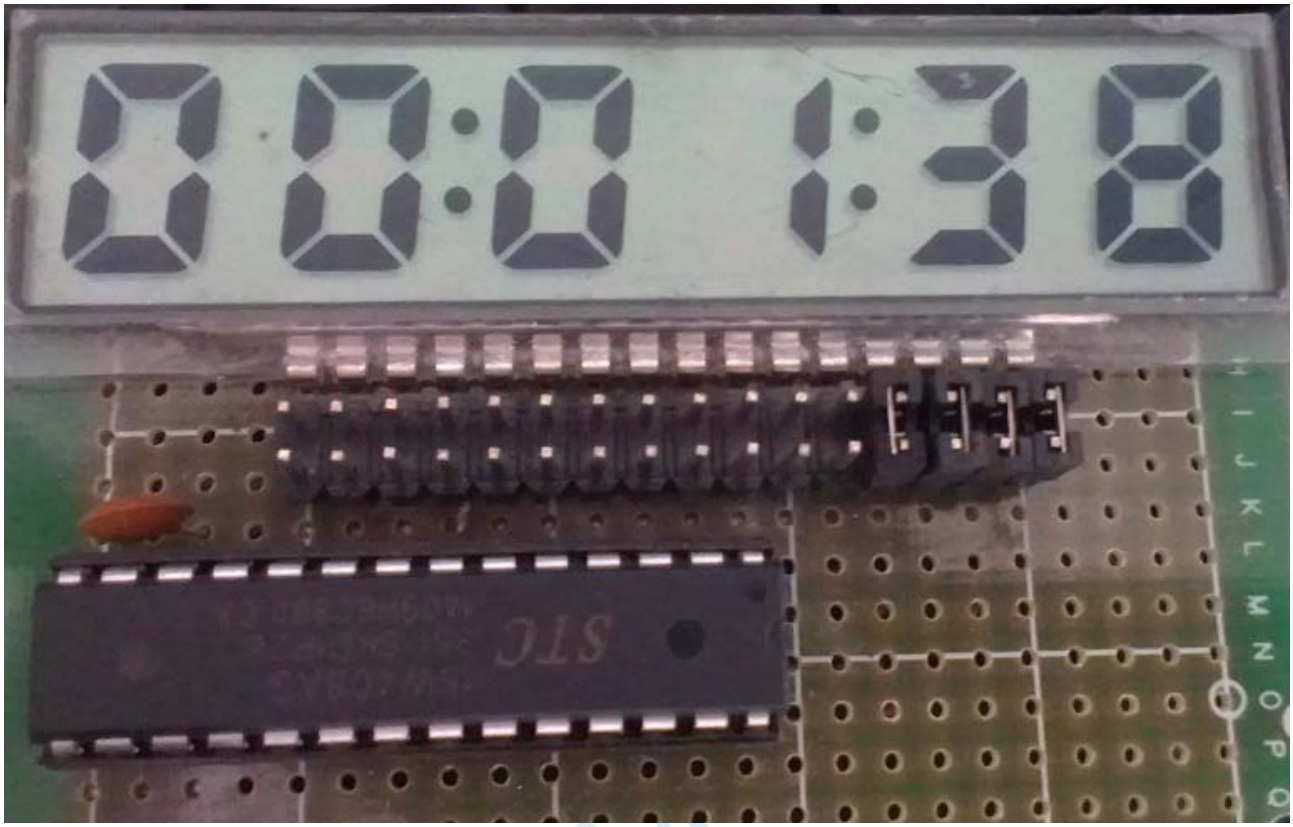
LCD真值表：

MCU PIN	P17	P16	P15	P14	P13	P12	P11	P10	P27	P26	P25	P24	P23	P22	P21	P20
LCD PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LCD PIN name	SEG11	SEG10	SEG9	SEG8	SEG7	SEG6	SEG5	SEG4	SEG3	SEG2	SEG1	SEG0	COM3	COM2	COM1	COM0
	--	1D	2:	2D	2.	3D	4:	4D	4.	5D	5.	6D	COM3			
	1E	1C	2E	2C	3E	3C	4E	4C	5E	5C	6E	6C		COM2		
	1G	1B	2G	2B	3G	3B	4G	4B	5G	5B	6G	6B			COM1	
	1F	1A	2F	2A	3F	3A	4F	4A	5F	5A	6F	6A				COM0

显存影射表：

	B7	B6	B5	B4	B3	B2	B1	B0
buff[0]:	--	1D	2:	2D	2.	3D	4:	4D
buff[1]:	1E	1C	2E	2C	3E	3C	4E	4C
buff[2]:	1G	1B	2G	2B	3G	3B	4G	4B
buff[3]:	1F	1A	2F	2A	3F	3A	4F	4A
buff[4]:	4.	5D	5.	6D	--	--	--	--
buff[5]:	5E	5C	6E	6C	--	--	--	--
buff[6]:	5G	5B	6G	6B	--	--	--	--
buff[7]:	5F	5A	6F	6A	--	--	--	--

图 7：驱动效果照片



本 LCD 扫描程序仅需要两个函数：

1、 LCD 段码扫描函数 void LCD_scan(void)

程序隔一定的时间调用这个函数，就会将 LCD 显示缓冲的内容显示到 LCD 上，全部扫描一次需要 8 个调用周期，调用间隔一般是 1~2ms，假如使用 1ms，则扫描周期就是 8ms，刷新率就是 125HZ。

2、 LCD 段码显示缓冲装载函数 void LCD_load(u8 n,u8 dat)

本函数用来将显示的数字或字符放在 LCD 显示缓冲中，比如 LCD_load(1,6)，就是要在第一个数字位置显示数字 6，支持显示 0~9，A~F，其它字符用户可以自己添加。

另外，用宏来显示、熄灭或闪烁冒号或小数点。

汇编代码

```
;用 STC8 系列测试 I/O 直接驱动段码 LCD(6 个 8 字 LCD, 1/4 Duty, 1/3 bias)。
;上电后显示一个时间(时分秒).

;*************************************************************
P0M1        DATA        0x93
P0M0        DATA        0x94
P1M1        DATA        0x91
P1M0        DATA        0x92
P2M1        DATA        0x95
P2M0        DATA        0x96
P3M1        DATA        0xB1
P3M0        DATA        0xB2
P4M1        DATA        0xB3
```

```
P4M0        DATA    0xB4
P5M1        DATA    0xC9
P5M0        DATA    0xC
P6M1        DATA    0xCB
P6M0        DATA    0xCC
P7M1        DATA    0xE1
P7M0        DATA    0xE2
AUXR        DATA    0x8E
INT_CLKO    DATA    0x8F
IE2         DATA    0xAF
P4          DATA    0xC0
T2H         DATA    0xD6
T2L         DATA    0xD7

;*******************************************************
DIS_BLACK   EQU     010H
DIS_        EQU     011H
DIS_A       EQU     00AH
DIS_B       EQU     00BH
DIS_C       EQU     00CH
DIS_D       EQU     00DH
DIS_E       EQU     00EH
DIS_F       EQU     00FH

B_2ms       BIT     20H.0           ;2ms 信号
B_Second    BIT     20H.1           ;秒信号
cnt_500ms   DATA    30H
second      DATA    31H
minute      DATA    32H
hour        DATA    33H
scan_index  DATA    34H

LCD_buff    DATA    40H             ;40H~47H

;*******************************************************
            ORG     0000H
            LJMP    F_Main

            ORG     000BH
            LJMP    F_Timer0_Interrupt

;*******************************************************
            ORG     0100H
F_Main:
            CLR     A
            MOV     P3M1, A         ;设置为准双向口
            MOV     P3M0, A
            MOV     P5M1, A         ;设置为准双向口
            MOV     P5M0, A

            MOV     P1M1, #0        ; segment 设置为推挽输出
            MOV     P1M0, #0ffh
            ANL     P2M1, #NOT 0f0h ; segment 设置为推挽输出
            ORL     P2M0, #0f0h
            ORL     P2M1, #00fH     ;全部COM 输出高阻, COM 为中点电压
            ANL     P2M0, #0f0H
            MOV     SP, #0D0H
            MOV     PSW, #0
            USING   0               ;选择第 0 组 R0~R7
```

STC MCU

```
;*************************************************************
                MOV         R2, #8
                MOV         R0, #LCD_buff
L_ClearLcdRam:
                MOV         @R0, #0
                INC         R0
                DJNZ        R2, L_ClearLcdRam

                LCALL       F_Timer0_init
                SETB        EA

;               ORL         LCD_buff, #020H         ;显示时分间隔:
;               ORL         LCD_buff, #002H         ;显示分秒间隔:

                MOV         hour, #12
                MOV         minute, #00
                MOV         second, #00
                LCALL       F_LoadRTC               ;显示时间

;*************************************************************
L_Main_Loop:
                JNB         B_2ms, L_Main_Loop      ;2ms 节拍到
                CLR         B_2ms

                INC         cnt_500ms
                MOV         A, cnt_500ms
                CJNE        A, #250, L_Main_Loop
                                                    ;500ms 到
                MOV         cnt_500ms, #0;

                XRL         LCD_buff, #020H         ;闪烁时分间隔:
                XRL         LCD_buff, #002H         ;闪烁分秒间隔:

                CPL         B_Second
                JNB         B_Second, L_Main_Loop

                INC         second
                MOV         A, second
                CJNE        A, #60, L_Main_Load
                MOV         second, #0              ; 1 分钟到
                INC         minute
                MOV         A, minute
                CJNE        A, #60, L_Main_Load
                MOV         minute, #0;
                INC         hour
                MOV         A, hour
                CJNE        A, #24, L_Main_Load
                MOV         hour, #0                ;24 小时到
L_Main_Load:
                LCALL       F_LoadRTC               ;显示时间
                LJMP        L_Main_Loop

;*************************************************************

F_Timer0_init:
                CLR         TR0                     ; 停止计数
                ANL         TMOD, #0f0H
                SETB        ET0                     ; 允许中断
```

```
        ORL     TMOD, #0                ; 工作模式, 0: 16 位自动重装
        ANL     INT_CLKO, #NOT 0x01     ; 不输出时钟
        ORL     AUXR, #0x80             ; 1T mode
        MOV     TH0, #HIGH (-22118)     ; 2ms
        MOV     TL0, #LOW  (-22118)     ;
        SETB    TR0                     ; 开始运行
        RET

;*************************************************************
F_Timer0_Interrupt:     ;Timer0 1ms 中断函数
        PUSH    PSW                     ;PSW 入栈
        PUSH    ACC                     ;ACC 入栈
        PUSH    AR0
        PUSH    AR7
        PUSH    DPH
        PUSH    DPL

        LCALL   F_LCD_scan
        SETB    B_2ms

        POP     DPL
        POP     DPH
        POP     AR7
        POP     AR0
        POP     ACC                     ;ACC 出栈
        POP     PSW                     ;PSW 出栈
        RETI

;******************** 显示时间 **************************
F_LoadRTC:
        MOV     R6, #1                  ;LCD_load(1,hour/10);
        MOV     A, hour
        MOV     B, #10
        DIV     AB
        MOV     R7, A
        LCALL   F_LCD_load              ;R6 为第几个数字，为1~6，R7 为要显示的数字

        MOV     R6, #2                  ;LCD_load(2,hour%10);
        MOV     A, hour
        MOV     B, #10
        DIV     AB
        MOV     R7, B
        LCALL   F_LCD_load              ;R6 为第几个数字，为1~6，R7 为要显示的数字

        MOV     R6, #3                  ;LCD_load(3,minute/10);
        MOV     A, minute
        MOV     B, #10
        DIV     AB
        MOV     R7, A
        LCALL   F_LCD_load              ;R6 为第几个数字，为1~6，R7 为要显示的数字

        MOV     R6, #4                  ;LCD_load(4,minute%10);
        MOV     A, minute
        MOV     B, #10
        DIV     AB
        MOV     R7, B
        LCALL   F_LCD_load              ;R6 为第几个数字，为1~6，R7 为要显示的数字

        MOV     R6, #5                  ;LCD_load(5,second/10);
```

```
                MOV     A, second
                MOV     B, #10
                DIV     AB
                MOV     R7, A
                LCALL   F_LCD_load          ;R6 为第几个数字，为 1~6，R7 为要显示的数字

                MOV     R6, #6              ;LCD_load(6,second%10);
                MOV     A, second
                MOV     B, #10
                DIV     AB
                MOV     R7, B
                LCALL   F_LCD_load          ;R6 为第几个数字，为 1~6，R7 为要显示的数字

                RET

;*********************************************************
T_COM:
                DB      008H, 004H, 002H, 001H

F_LCD_scan:
                MOV     A, scan_index       ;j = scan_index >> 1;
                CLR     C
                RRC     A
                MOV     R7, A               ;R7 = j
                ADD     A, #LCD_buff
                MOV     R0, A               ;R0 = LCD_buff[j]
                ORL     P2M1, #00fH         ;全部 COM 输出高阻, COM 为中点电压
                ANL     P2M0, #0f0H

                MOV     A, scan_index
                JNB     ACC.0, L_LCD_Scan2  ;if(scan_index & 1)    //反相扫描
                MOV     A, @R0              ;P1 = ~LCD_buff[j];
                CPL     A
                MOV     P1, A
                MOV     A, R0               ;P2 = ~(LCD_buff[j|4] & 0xf0);
                ADD     A, #4
                MOV     R0, A
                MOV     A, @R0
                ANL     A, #0f0H
                CPL     A
                MOV     P2, A
                SJMP    L_LCD_Scan3

L_LCD_Scan2:                                ;正相扫描
                MOV     A, @R0              ;P1 = LCD_buff[j];
                MOV     P1, A
                MOV     A, R0               ;P2 = (LCD_buff[j|4] & 0xf0);
                ADD     A, #4
                MOV     R0, A
                MOV     A, @R0
                ANL     A, #0f0H
                MOV     P2, A

L_LCD_Scan3:
                MOV     DPTR, #T_COM        ;某个 COM 设置为推挽输出
                MOV     A, R7
                MOVC    A, @A+DPTR
                ORL     P2M0, A
                CPL     A
```

```
                ANL         P2M1, A

                INC         scan_index              ;if(++scan_index == 8)      scan_index = 0;
                MOV         A, scan_index
                CJNE        A, #8, L_QuitLcdScan
                MOV         scan_index, #0

L_QuitLcdScan:
                RET

;******************** 标准字库 ***************************
T_Display:
;               0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
DB              03FH,006H,05BH,04FH,066H,06DH,07DH,007H,07FH,06FH,077H,07CH,039H,05EH,079H,071H
;               black  -
DB              000H,040H

;******************** 对第1~6 数字装载显示函数 算法简单 ****************************
F_LCD_load:                                         ;R6 为第几个数字，为1~6，R7 为要显示的数字
                MOV         DPTR, #T_Display        ;i = t_display[dat];
                MOV         A, R7
                MOVC        A, @A+DPTR
                MOV         B, A                    ;要显示的数字

                MOV         A, R6
                CJNE        A, #1, L_NotLoadChar1
                MOV         R0,                     #LCD_buff
                MOV         A, @R0
                MOV         C, B.3                  ;D
                MOV         ACC.6, C
                MOV         @R0, A

                INC         R0
                MOV         A, @R0
                MOV         C, B.2                  ;C
                MOV         ACC.6, C
                MOV         C, B.4                  ;E
                MOV         ACC.7, C
                MOV         @R0, A

                INC         R0
                MOV         A, @R0
                MOV         C, B.1                  ;B
                MOV         ACC.6, C
                MOV         C, B.6                  ;G
                MOV         ACC.7, C
                MOV         @R0, A

                INC         R0
                MOV         A, @R0
                MOV         C, B.0                  ;A
                MOV         ACC.6, C
                MOV         C, B.5                  ;F
                MOV         ACC.7, C
                MOV         @R0, A
                RET

L_NotLoadChar1:
                CJNE        A, #2, L_NotLoadChar2
```

STC MCU

```
        MOV     R0,#LCD_buff
        MOV     A, @R0
        MOV     C, B.3          ;D
        MOV     ACC.4, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.2          ;C
        MOV     ACC.4, C
        MOV     C, B.4          ;E
        MOV     ACC.5, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.1          ;B
        MOV     ACC.4, C
        MOV     C, B.6          ;G
        MOV     ACC.5, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.0          ;A
        MOV     ACC.4, C
        MOV     C, B.5          ;F
        MOV     ACC.5, C
        MOV     @R0, A
        RET

L_NotLoadChar2:
        CJNE    A, #3, L_NotLoadChar3
        MOV     R0,#LCD_buff
        MOV     A, @R0
        MOV     C, B.3          ;D
        MOV     ACC.2, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.2          ;C
        MOV     ACC.2, C
        MOV     C, B.4          ;E
        MOV     ACC.3, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.1          ;B
        MOV     ACC.2, C
        MOV     C, B.6          ;G
        MOV     ACC.3, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.0          ;A
        MOV     ACC.2, C
```

STC MCU

```
        MOV     C, B.5          ;F
        MOV     ACC.3, C
        MOV     @R0, A
        RET

L_NotLoadChar3:
        CJNE    A, #4, L_NotLoadChar4
        MOV     R0,#LCD_buff
        MOV     A, @R0
        MOV     C, B.3          ;D
        MOV     ACC.0, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.2          ;C
        MOV     ACC.0, C
        MOV     C, B.4          ;E
        MOV     ACC.1, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.1          ;B
        MOV     ACC.0, C
        MOV     C, B.6          ;G
        MOV     ACC.1, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.0          ;A
        MOV     ACC.0, C
        MOV     C, B.5          ;F
        MOV     ACC.1, C
        MOV     @R0, A
        RET

L_NotLoadChar4:
        CJNE    A, #5, L_NotLoadChar5
        MOV     R0,#LCD_buff+4
        MOV     A, @R0
        MOV     C, B.3          ;D
        MOV     ACC.6, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.2          ;C
        MOV     ACC.6, C
        MOV     C, B.4          ;E
        MOV     ACC.7, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.1          ;B
        MOV     ACC.6, C
        MOV     C, B.6          ;G
```

```
            MOV     ACC.7, C
            MOV     @R0, A

            INC     R0
            MOV     A, @R0
            MOV     C, B.0          ;A
            MOV     ACC.6, C
            MOV     C, B.5          ;F
            MOV     ACC.7, C
            MOV     @R0, A
            RET

L_NotLoadChar5:
            CJNE    A, #6, L_NotLoadChar6
            MOV     R0,#LCD_buff+4
            MOV     A, @R0
            MOV     C, B.3          ;D
            MOV     ACC.4, C
            MOV     @R0, A

            INC     R0
            MOV     A, @R0
            MOV     C, B.2          ;C
            MOV     ACC.4, C
            MOV     C, B.4          ;E
            MOV     ACC.5, C
            MOV     @R0, A

            INC     R0
            MOV     A, @R0
            MOV     C, B.1          ;B
            MOV     ACC.4, C
            MOV     C, B.6          ;G
            MOV     ACC.5, C
            MOV     @R0, A

            INC     R0
            MOV     A, @R0
            MOV     C, B.0          ;A
            MOV     ACC.4, C
            MOV     C, B.5          ;F
            MOV     ACC.5, C
            MOV     @R0, A
            RET
L_NotLoadChar6:
            RET

            E                       N                       D
```

C 语言代码

```
/*****************功能说明******************
用 STC15 系列测试 I/O 直接驱动段码 LCD(6 个 8 字 LCD, 1/4 Dutys, 1/3 bias)。
上电后显示一个时间(时分秒).
P3.2 对地接一个开关,用来进入睡眠或唤醒.
*********************************************/

#include "reg51.h"
#include "intrins.h"
```

```
typedef     unsigned char       u8;
typedef     unsigned int        u16;
typedef     unsigned long       u32;

sfr AUXR = 0x8e;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;

/*********************本地常量声明**************************/
#define MAIN_Fosc           11059200L                   //定义主时钟

#define DIS_BLACK           0x10
#define DIS_                0x11
#define DIS_A               0x0A
#define DIS_B               0x0B
#define DIS_C               0x0C
#define DIS_D               0x0D
#define DIS_E               0x0E
#define DIS_F               0x0F

#define LCD_SET_DP2         LCD_buff[0] |=    0x08
#define LCD_CLR_DP2         LCD_buff[0] &= ~0x08
#define LCD_FLASH_DP2       LCD_buff[0] ^=    0x08

#define LCD_SET_DP4         LCD_buff[4] |=    0x80
#define LCD_CLR_DP4         LCD_buff[4] &= ~0x80
#define LCD_FLASH_DP4       LCD_buff[4] ^=    0x80

#define LCD_SET_2M          LCD_buff[0] |=    0x20
#define LCD_CLR_2M          LCD_buff[0] &= ~0x20
#define LCD_FLASH_2M        LCD_buff[0] ^=    0x20

#define LCD_SET_4M          LCD_buff[0] |=    0x02
#define LCD_CLR_4M          LCD_buff[0] &= ~0x02
#define LCD_FLASH_4M        LCD_buff[0] ^=    0x02

#define LCD_SET_DP5         LCD_buff[4] |=    0x20
#define LCD_CLR_DP5         LCD_buff[4] &= ~0x20
#define LCD_FLASH_DP5       LCD_buff[4] ^=    0x20

#define P1n_standard(bitn)      P1M1 &= ~(bitn), P1M0 &= ~(bitn)
#define P1n_push_pull(bitn)     P1M1 &= ~(bitn), P1M0 |= (bitn)
#define P1n_pure_input(bitn)    P1M1 |= (bitn), P1M0 &= ~(bitn)
#define P1n_open_drain(bitn)    P1M1 |= (bitn), P1M0 |= (bitn)

#define P2n_standard(bitn)      P2M1 &= ~(bitn), P2M0 &= ~(bitn)
#define P2n_push_pull(bitn)     P2M1 &= ~(bitn), P2M0 |= (bitn)
#define P2n_pure_input(bitn)    P2M1 |= (bitn), P2M0 &= ~(bitn)
#define P2n_open_drain(bitn)    P2M1 |= (bitn), P2M0 |= (bitn)

/*********************本地变量声明**************************/
u8 cnt_500ms;
u8 second,minute,hour;
bit B_Second;
bit B_2ms;
u8 LCD_buff[8];
```

```
u8 scan_index;

/********************本地函数声明*************************/
void LCD_load(u8 n,u8 dat);
void LCD_scan(void);
void LoadRTC(void);
void delay_ms(u8 ms);

/********************主函数*************************/
void main(void)
{
    u8    i;

    AUXR = 0x80;
    TMOD = 0x00;
    TL0 = (65536 - (MAIN_Fosc / 500));
    TH0 = (65536 - (MAIN_Fosc / 500)) >> 8;
    TR0 = 1;
    ET0 = 1;
    EA = 1;

                                                    //初始化 LCD 显存
    for(i=0; i<8; i++) LCD_buff[i] = 0;
    P2n_push_pull(0xf0);
    P1n_push_pull(0xff);                            //segment 设置为推挽输出

    LCD_SET_2M;                                     //显示时分间隔:
    LCD_SET_4M;                                     //显示分秒间隔:
    LoadRTC();                                      //显示时间

    while (1)
    {
        PCON |= 0x01;                               //进入空闲模式，由 Timer0 2ms 唤醒退出
        _nop_();
        _nop_();
        _nop_();

        if(B_2ms)                                   //2ms 节拍到
        {
            B_2ms = 0;

            if(++cnt_500ms >= 250)                  //500ms 到
            {
                cnt_500ms = 0;
            //  LCD_FLASH_2M;                       //闪烁时分间隔:
            //  LCD_FLASH_4M;                       //闪烁分秒间隔:

                B_Second = ~B_Second;
                if(B_Second)
                {
                    if(++second >= 60)              //1 分钟到
                    {
                        second = 0;
                        if(++minute >= 60)          //1 小时到
                        {
                            minute = 0;
                            if(++hour >= 24) hour = 0;  //24 小时到
                        }
                    }
```

```
                LoadRTC();                          //显示时间
            }
        }

        if(!INT0)                                   //键按下，准备睡眠
        {
            LCD_CLR_2M;                             //显示时分间隔:
            LCD_CLR_4M;                             //显示分秒间隔:
            LCD_load(1,DIS_BLACK);
            LCD_load(2,DIS_BLACK);
            LCD_load(3,0);
            LCD_load(4,0x0F);
            LCD_load(5,0x0F);
            LCD_load(6,DIS_BLACK);

            while(!INT0) delay_ms(10);              //等待释放按键
            delay_ms(50);
            while(!INT0) delay_ms(10);              //再次等待释放按键

            TR0 = 0;                                //关闭定时器
            IE0 = 0;                                //外中断 0 标志位
            EX0 = 1;                                //INT0 Enable
            IT0 = 1;                                //INT0  下降沿中断

            P1n_push_pull(0xff);                    //com 和 seg 全部输出 0
            P2n_push_pull(0xff);
            P1 = 0;
            P2 = 0;

            PCON |= 0x02;                           //Sleep
            _nop_();
            _nop_();
            _nop_();

            LCD_SET_2M;                             //显示时分间隔:
            LCD_SET_4M;                             //显示分秒间隔:
            LoadRTC();                              //显示时间
            TR0 = 1;                                //打开定时器
            while(!INT0) delay_ms(10);              //等待释放按键
            delay_ms(50);
            while(!INT0) delay_ms(10);              //再次等待释放按键
        }
    }
}
}

/********************延时函数*************************/
void delay_ms(u8 ms)
{
    unsigned int i;
    do{
        i = MAIN_Fosc / 13000;
        while(--i);                                 //14T per loop
    }while(--ms);
}

/********************* Timer0 中断函数************************/
void timer0_int (void) interrupt 1
{
```

```
    LCD_scan();
    B_2ms = 1;
}

/********************* INT0 中断函数 ************************/
void INT0_int (void) interrupt 0
{
    EX0 = 0;
    IE0  = 0;
}

/******************* LCD 段码扫描函数 *************************/
void  LCD_scan(void)                                    //5us @22.1184MHZ
{
    u8 code T_COM[4]={0x08,0x04,0x02,0x01};
    u8 j;

    j = scan_index >> 1;
    P2n_pure_input(0x0f);                               //全部COM 输出高阻, COM 为中点电压
    if(scan_index & 1)                                  //反相扫描
    {
        P1 = ~LCD_buff[j];
        P2 = ~(LCD_buff[j|4] & 0xf0);
    }
    else                                                //正相扫描
    {
        P1 = LCD_buff[j];
        P2 = LCD_buff[j|4] & 0xf0;
    }
    P2n_push_pull(T_COM[j]);                            //某个COM 设置为推挽输出
    if(++scan_index >= 8) scan_index = 0;
}

/******************* 对第1~6 数字装载显示函数 *************************/
void  LCD_load(u8 n, u8 dat)                            //n 为第几个数字，dat 为要显示的数字
{
    u8 code t_display[]={                               //标准字库
        // 0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
        0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71,
        //black  -
        0x00,0x40
    };
    u8 code T_LCD_mask[4]   = {~0xc0,~0x30,~0x0c,~0x03};
    u8 code T_LCD_mask4[4] = {~0x40,~0x10,~0x04,~0x01};
    u8 i,k;
    u8 *p;

    if((n == 0) || (n > 6)) return;
    i = t_display[dat];

    if(n <= 4)                                          //1~4
    {
        n--;
        p = LCD_buff;
    }
    else
    {
        n = n - 5;
        p = &LCD_buff[4];
```

```
    }

    k = 0;
    if(i & 0x08) k |= 0x40;                    //D
    *p = (*p & T_LCD_mask4[n]) | (k>>2*n);
    p++;

    k = 0;
    if(i & 0x04) k |= 0x40;                    //C
    if(i & 0x10) k |= 0x80;                    //E
    *p = (*p & T_LCD_mask[n]) | (k>>2*n);
    p++;

    k = 0;
    if(i & 0x02) k |= 0x40;                    //B
    if(i & 0x40) k |= 0x80;                    //G
    *p = (*p & T_LCD_mask[n]) | (k>>2*n);
    p++;

    k = 0;
    if(i & 0x01) k |= 0x40;                    //A
    if(i & 0x20) k |= 0x80;                    //F
    *p = (*p & T_LCD_mask[n]) | (k>>2*n);
}

/*********************显示时间 **************************/
void LoadRTC(void)
{
    LCD_load(1,hour/10);
    LCD_load(2,hour%10);
    LCD_load(3,minute/10);
    LCD_load(4,minute%10);
    LCD_load(5,second/10);
    LCD_load(6,second%10);
}
```

STC MCU

10 指令系统

助记符		指令说明	字节	时钟
ADD	A,Rn	寄存器内容加到累加器	1	1
ADD	A,direct	直接地址单元的数据加到累加器	2	1
ADD	A,@Ri	间接地址单元的数据加到累加器	1	1
ADD	A,#data	立即数加到累加器	2	1
ADDC	A,Rn	寄存器带进位加到累加器	1	1
ADDC	A,direct	直接地址单元的数据带进位加到累加器	2	1
ADDC	A,@Ri	间接地址单元的数据带进位加到累加器	1	1
ADDC	A,#data	立即数带进位加到累加器	2	1
SUBB	A,Rn	累加器带借位减寄存器内容	1	1
SUBB	A,direct	累加器带借位减直接地址单元的内容	2	1
SUBB	A,@Ri	累加器带借位减间接地址单元的内容	1	1
SUBB	A,#data	累加器带借位减立即数	2	1
INC	A	累加器加1	1	1
INC	Rn	寄存器加1	1	1
INC	direct	直接地址单元加1	2	1
INC	@Ri	间接地址单元加1	1	1
DEC	A	累加器减1	1	1
DEC	Rn	寄存器减1	1	1
DEC	direct	直接地址单元减1	2	1
DEC	@Ri	间接地址单元减1	1	1
INC	DPTR	地址寄存器DPTR加1	1	1
MUL	AB	A乘以B, B存放高字节, A存放低字节	1	2
DIV	AB	A除以B, B存放余数, A存放商	1	6
DA	A	累加器十进制调整	1	3
ANL	A,Rn	累加器与寄存器相与	1	1
ANL	A,direct	累加器与直接地址单元相与	2	1
ANL	A,@Ri	累加器与间接地址单元相与	1	1
ANL	A,#data	累加器与立即数相与	2	1
ANL	direct,A	直接地址单元与累加器相与	2	1
ANL	direct,#data	直接地址单元与立即数相与	3	1
ORL	A,Rn	累加器与寄存器相或	1	1
ORL	A,direct	累加器与直接地址单元相或	2	1
ORL	A,@Ri	累加器与间接地址单元相或	1	1

ORL	A,#data	累加器与立即数相或	2	1
ORL	direct,A	直接地址单元与累加器相或	2	1
ORL	direct,#data	直接地址单元与立即数相或	3	1
XRL	A,Rn	累加器与寄存器相异或	1	1
XRL	A,direct	累加器与直接地址单元相异或	2	1
XRL	A,@Ri	累加器与间接地址单元相异或	1	1
XRL	A,#data	累加器与立即数相异或	2	1
XRL	direct,A	直接地址单元与累加器相异或	2	1
XRL	direct,#data	直接地址单元与立即数相异或	3	1
CLR	A	累加器清0	1	1
CPL	A	累加器取反	1	1
RL	A	累加器循环左移	1	1
RLC	A	累加器带进位循环左移	1	1
RR	A	累加器循环右移	1	1
RRC	A	累加器带进位循环右移	1	1
SWAP	A	累加器高低半字节交换	1	1
CLR	C	清零进位位	1	1
CLR	bit	清0直接地址位	2	1
SETB	C	置1进位位	1	1
SETB	bit	置1直接地址位	2	1
CPL	C	进位位求反	1	1
CPL	bit	直接地址位求反	2	1
ANL	C,bit	进位位和直接地址位相与	2	1
ANL	C,/bit	进位位和直接地址位的反码相与	2	1
ORL	C,bit	进位位和直接地址位相或	2	1
ORL	C,/bit	进位位和直接地址位的反码相或	2	1
MOV	C,bit	直接地址位送入进位位	2	1
MOV	bit,C	进位位送入直接地址位	2	1
MOV	A,Rn	寄存器内容送入累加器	1	1
MOV	A,direct	直接地址单元中的数据送入累加器	2	1
MOV	A,@Ri	间接地址中的数据送入累加器	1	1
MOV	A,#data	立即数送入累加器	2	1
MOV	Rn,A	累加器内容送入寄存器	1	1
MOV	Rn,direct	直接地址单元中的数据送入寄存器	2	1
MOV	Rn,#data	立即数送入寄存器	2	1
MOV	direct,A	累加器内容送入直接地址单元	2	1
MOV	direct,Rn	寄存器内容送入直接地址单元	2	1

MOV	direct,direct	直接地址单元中的数据送入另一个直接地址单元	3	1
MOV	direct,@Ri	间接地址中的数据送入直接地址单元	2	1
MOV	direct,#data	立即数送入直接地址单元	3	1
MOV	@Ri,A	累加器内容送间接地址单元	1	1
MOV	@Ri,direct	直接地址单元数据送入间接地址单元	2	1
MOV	@Ri,#data	立即数送入间接地址单元	2	1
MOV	DPTR,#data16	16位立即数送入数据指针	3	1
MOVC	A,@A+DPTR	以DPTR为基地址变址寻址单元中的数据送入累加器	1	4
MOVC	A,@A+PC	以PC为基地址变址寻址单元中的数据送入累加器	1	3
MOVX	A,@Ri	扩展地址(8位地址)的内容送入累加器A中	1	3[1]
MOVX	A,@DPTR	扩展RAM(16位地址)的内容送入累加器A中	1	2[1]
MOVX	@Ri,A	将累加器A的内容送入扩展RAM(8位地址)中	1	3[1]
MOVX	@DPTR,A	将累加器A的内容送入扩展RAM(16位地址)中	1	2[1]
PUSH	direct	直接地址单元中的数据压入堆栈	2	1
POP	direct	栈底数据弹出送入直接地址单元	2	1
XCH	A,Rn	寄存器与累加器交换	1	1
XCH	A,direct	直接地址单元与累加器交换	2	1
XCH	A,@Ri	间接地址与累加器交换	1	1
XCHD	A,@Ri	间接地址的低半字节与累加器交换	1	1
ACALL	addr11	短调用子程序	2	3
LCALL	addr16	长调用子程序	3	3
RET		子程序返回	1	3
RETI		中断返回	1	3
AJMP	addr11	短跳转	2	3
LJMP	addr16	长跳转	3	3
SJMP	rel	相对跳转	2	3
JMP	@A+DPTR	相对于DPTR的间接跳转	1	4
JZ	rel	累加器为零跳转	2	1/3[2]
JNZ	rel	累加器非零跳转	2	1/3[2]
JC	rel	进位位为1跳转	2	1/3[2]
JNC	rel	进位位为0跳转	2	1/3[2]
JB	bit,rel	直接地址位为1则跳转	3	1/3[2]
JNB	bit,rel	直接地址位为0则跳转	3	1/3[2]
JBC	bit,rel	直接地址位为1则跳转, 该位清0	3	1/3[2]
CJNE	A,direct,rel	累加器与直接地址单元不相等跳转	3	2/3[3]
CJNE	A,#data,rel	累加器与立即数不相等跳转	3	1/3[2]
CJNE	Rn,#data,rel	寄存器与立即数不相等跳转	3	2/3[3]

CJNE @Ri,#data,rel	间接地址单元与立即数不相等跳转	3	2/3[3]
DJNZ Rn,rel	寄存器减1后非零跳转	2	2/3[3]
DJNZ direct,rel	直接地址单元减1后非零跳转	3	2/3[3]
NOP	空操作	1	1

[1]:访问外部扩展 RAM 时，指令的执行周期与寄存器 BUS_SPEED 中的 SPEED[2:0]位有关

[2]:对于条件跳转语句的执行时间会依据条件是否满足而不同。当条件不满足时，不会发生跳转而继续执行下一条指令，此时条件跳转语句的执行时间为 1 个时钟；当条件满足时，则会发生跳转，此时条件跳转语句的执行时间为 3 个时钟。

[3]:对于条件跳转语句的执行时间会依据条件是否满足而不同。当条件不满足时，不会发生跳转而继续执行下一条指令，此时条件跳转语句的执行时间为 2 个时钟；当条件满足时，则会发生跳转，此时条件跳转语句的执行时间为 3 个时钟。

STC MCU

11 中断系统

中断系统是为使 CPU 具有对外界紧急事件的实时处理能力而设置的。

当中央处理机 CPU 正在处理某件事的时候外界发生了紧急事件请求，要求 CPU 暂停当前的工作，转而去处理这个紧急事件，处理完以后，再回到原来被中断的地方，继续原来的工作，这样的过程称为中断。实现这种功能的部件称为中断系统，请示 CPU 中断的请求源称为中断源。微型机的中断系统一般允许多个中断源，当几个中断源同时向 CPU 请求中断，要求为它服务的时候，这就存在 CPU 优先响应哪一个中断源请求的问题。通常根据中断源的轻重缓急排队，优先处理最紧急事件的中断请求源，即规定每一个中断源有一个优先级别。CPU 总是先响应优先级别最高的中断请求。

当 CPU 正在处理一个中断源请求的时候（执行相应的中断服务程序），发生了另外一个优先级比它还高的中断源请求。如果 CPU 能够暂停对原来中断源的服务程序，转而去处理优先级更高的中断请求源，处理完以后，再回到原低级中断服务程序，这样的过程称为中断嵌套。这样的中断系统称为多级中断系统，没有中断嵌套功能的中断系统称为单级中断系统。

用户可以用关总中断允许位（EA/IE.7）或相应中断的允许位屏蔽相应的中断请求，也可以用打开相应的中断允许位来使 CPU 响应相应的中断申请，每一个中断源可以用软件独立地控制为开中断或关中断状态，部分中断的优先级别均可用软件设置。高优先级的中断请求可以打断低优先级的中断，反之，低优先级的中断请求不可以打断高优先级的中断。当两个相同优先级的中断同时产生时，将由查询次序来决定系统先响应哪个中断。

11.1 STC8G系列中断源

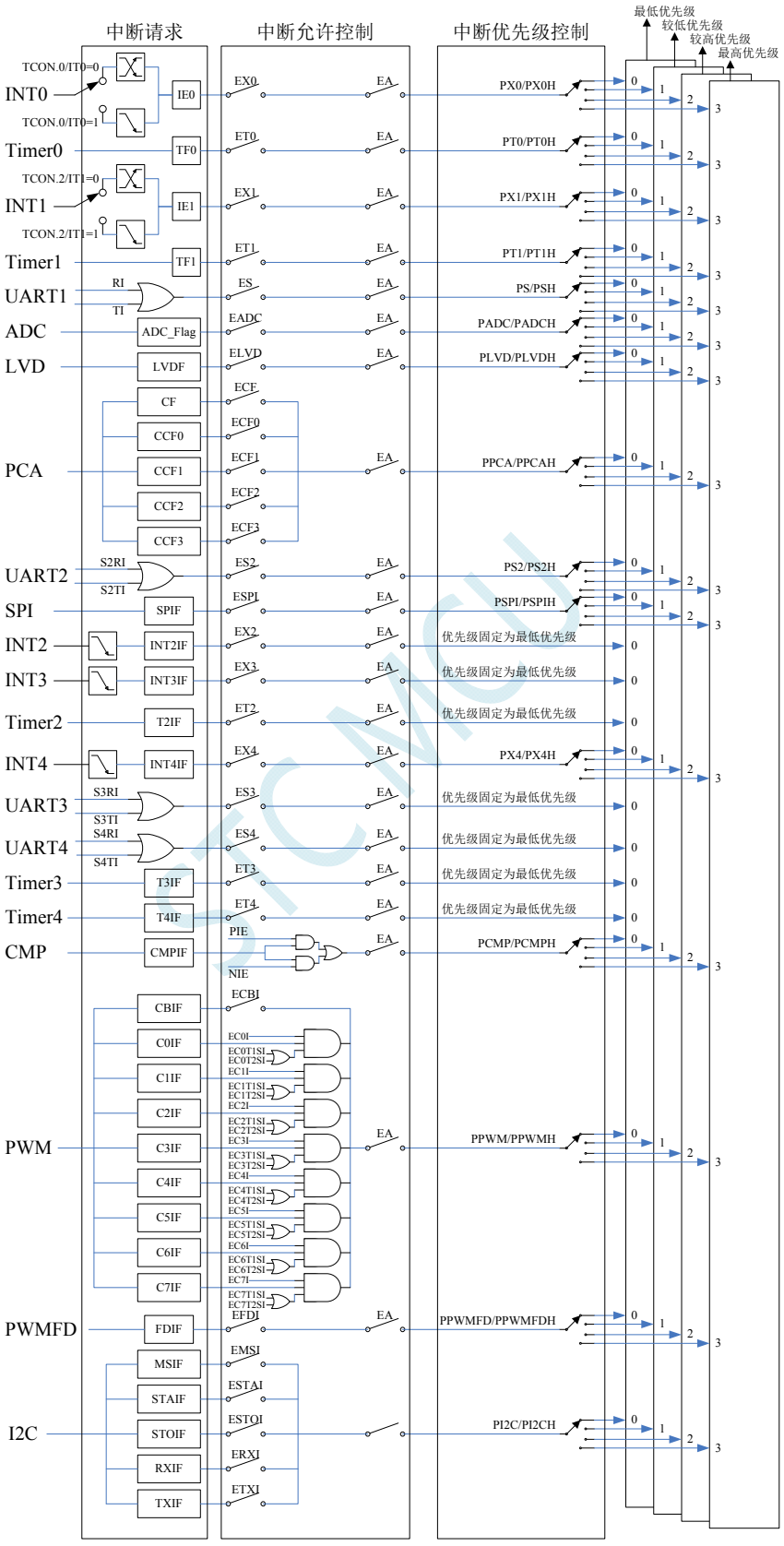
下表中√表示对应的系列有相应的中断源

中断源	STC8A8K64S4A12 系列	STC8A4K64S2A12 系列	STC8F2K64S4 系列	STC8F2K64S2 系列	STC8F1K08S2 系列	STC8F1K08 系列
外部中断 0 中断（INT0）	√	√	√	√	√	√
定时器 0 中断（Timer0）	√	√	√	√	√	√
外部中断 1 中断（INT1）	√	√	√	√	√	√
定时器 1 中断（Timer1）	√	√	√	√	√	√
串口 1 中断（UART1）	√	√	√	√	√	√
模数转换中断（ADC）	√	√				
低压检测中断（LVD）	√	√	√	√	√	√
捕获中断（CCP/PCA/PWM）	√	√				
串口 2 中断（UART2）	√	√	√	√	√	
串行外设接口中断（SPI）	√	√	√	√	√	√
外部中断 2 中断（INT2）	√	√	√	√	√	
外部中断 3 中断（INT3）	√	√	√	√	√	
定时器 2 中断（Timer2）	√	√	√	√	√	√
外部中断 4 中断（INT4）	√	√	√	√	√	√
串口 3 中断（UART3）	√		√			

串口 4 中断（UART4）	√		√			
定时器 3 中断（Timer3）	√	√	√	√		
定时器 4 中断（Timer4）	√	√	√	√		
比较器中断（CMP）	√	√	√	√	√	
增强型 PWM 中断	√	√				
PWM 异常检测中断（PWMFD）	√	√				
I2C 总线中断	√	√	√	√	√	√

STC MCU

11.2 STC8A/8F系列中断结构图



11.3 STC8A/8F系列中断列表

中断源	中断向量	次序	优先级设置	优先级	中断请求位	中断允许位
INT0	0003H	0	PX0PX0H	0/1/2/3	IE0	EX0
Timer0	000BH	1	PT0,PT0H	0/1/2/3	TF0	ET0
INT1	0013H	2	PX1,PX1H	0/1/2/3	IE1	EX1
Timer1	001BH	3	PT1,PT1H	0/1/2/3	TF1	ET1
UART1	0023H	4	PS,PSH	0/1/2/3	RI || TI	ES
ADC	002BH	5	PADC,PADCH	0/1/2/3	ADC_FLAG	EADC
LVD	0033H	6	PLVD,PLVDH	0/1/2/3	LVDF	ELVD
PCA	003BH	7	PPCA,PPCAH	0/1/2/3	CF	ECF
					CCF0	ECCF0
					CCF1	ECCF1
					CCF2	ECCF2
					CCF3	ECCF3
UART2	0043H	8	PS2,PS2H	0/1/2/3	S2RI || S2TI	ES2
SPI	004BH	9	PSPI,PSPIH	0/1/2/3	SPIF	ESPI
INT2	0053H	10		0	INT2IF	EX2
INT3	005BH	11		0	INT3IF	EX3
Timer2	0063H	12		0	T2IF	ET2
INT4	0083H	16	PX4,PX4H	0/1/2/3	INT4IF	EX4
UART3	008BH	17	~~PS3,PS3H~~	~~0/1/2/3~~	S3RI || S3TI	ES3
UART4	0093H	18	~~PS4,PS4H~~	~~0/1/2/3~~	S4RI || S4TI	ES4
Timer3	009BH	19		0	T3IF	ET3
Timer4	00A3H	20		0	T4IF	ET4
CMP	00ABH	21	PCMP,PCMPH	0/1/2/3	CMPIF	PIE|NIE

STC MCU

中断源	中断向量	次序	优先级设置	优先级	中断请求位	中断允许位
PWM	00B3H	22	PPWM,PPWMH	0/1/2/3	CBIF	ECBI
					C0IF	EC0I && EC0T1SI
						EC0I && EC0T2SI
					C1IF	EC1I && EC1T1SI
						EC1I && EC1T2SI
					C2IF	EC2I && EC2T1SI
						EC2I && EC2T2SI
					C3IF	EC3I && EC3T1SI
						EC3I && EC3T2SI
					C4IF	EC4I && EC4T1SI
						EC4I && EC4T2SI
					C5IF	EC5I && EC5T1SI
						EC5I && EC5T2SI
					C6IF	EC6I && EC6T1SI
						EC6I && EC6T2SI
					C7IF	EC7I && EC7T1SI
						EC7I && EC7T2SI
PWMFD	00BBH	23	PPWMFD,PPWMFDH	0/1/2/3	FDIF	EFDI
I2C	00C3H	24	PI2C,PI2CH	0/1/2/3	MSIF	EMSI
					STAIF	ESTAI
					RXIF	ERXI
					TXIF	ETXI
					STOIF	ESTOI

在 C 语言中声明中断服务程序

```
void  INT0_Routine(void)      interrupt  0;
void  TM0_Rountine(void)      interrupt  1;
void  INT1_Routine(void)      interrupt  2;
void  TM1_Rountine(void)      interrupt  3;
void  UART1_Routine(void)     interrupt  4;
void  ADC_Routine(void)       interrupt  5;
void  LVD_Routine(void)       interrupt  6;
void  PCA_Routine(void)       interrupt  7;
void  UART2_Routine(void)     interrupt  8;
void  SPI_Routine(void)       interrupt  9;
void  INT2_Routine(void)      interrupt  10;
void  INT3_Routine(void)      interrupt  11;
```

```
void    TM2_Routine(void)       interrupt  12;
void    INT4_Routine(void)      interrupt  16;
void    UART3_Routine(void)     interrupt  17;
void    UART4_Routine(void)     interrupt  18;
void    TM3_Routine(void)       interrupt  19;
void    TM4_Routine(void)       interrupt  20;
void    CMP_Routine(void)       interrupt  21;
void    PWM_Routine(void)       interrupt  22;
void    PWMFD_Routine(void)     interrupt  23;
void    I2C_Routine(void)       interrupt  24;
```

STC MCU

11.4 中断相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IE	中断允许寄存器	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
IE2	中断允许寄存器 2	AFH		ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000,0000
INTCLKO	中断与时钟输出控制寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
IP	中断优先级控制寄存器	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000,0000
IPH	高中断优先级控制寄存器	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	0000,0000
IP2	中断优先级控制寄存器 2	B5H		PI2C	PCMP	PX4	PPWMFD	PPWM	PSPI	PS2	x000,0000
IP2H	高中断优先级控制寄存器 2	B6H		PI2CH	PCMPH	PX4H	PPWMFDH	PPWMH	PSPIH	PS2H	x000,0000
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
AUXINTIF	扩展外部中断标志寄存器	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S3CON	串口 3 控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S4CON	串口 4 控制寄存器	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]				000x,0000
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
CCON	PCA 控制寄存器	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0	00xx,0000
CMOD	PCA 模式寄存器	D9H	CIDL	-	-	-	CPS[2:0]			ECF	0xxx,0000
CCAPM0	PCA 模块 0 模式控制寄存器	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA 模块 1 模式控制寄存器	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA 模块 2 模式控制寄存器	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
CCAPM3	PCA 模块 3 模式控制寄存器	DDH	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3	x000,0000
CMPCR1	比较器控制寄存器 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
PWMCFG	增强型 PWM 配置寄存器	F1H	CBIF	ETADC	-	-	-	-	-	-	00xx,xxxx
PWMCR	PWM 控制寄存器	FEH	ENPWM	ECBI	-	-	-	-	-	-	00xx,xxxx
PWMIF	增强型 PWM 中断标志寄存器	F6H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWMFDCR	PWM 异常检测控制寄存器	F7H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWM0CR	PWM0 控制寄存器	FF04H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI	00x0,0000
PWM1CR	PWM1 控制寄存器	FF14H	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI	00x0,0000
PWM2CR	PWM2 控制寄存器	FF24H	ENC2O	C2INI	-	C2_S[1:0]		EC2I	EC2T2SI	EC2T1SI	00x0,0000
PWM3CR	PWM3 控制寄存器	FF34H	ENC3O	C3INI	-	C3_S[1:0]		EC3I	EC3T2SI	EC3T1SI	00x0,0000
PWM4CR	PWM4 控制寄存器	FF44H	ENC4O	C4INI	-	C4_S[1:0]		EC4I	EC4T2SI	EC4T1SI	00x0,0000
PWM5CR	PWM5 控制寄存器	FF54H	ENC5O	C5INI	-	C5_S[1:0]		EC5I	EC5T2SI	EC5T1SI	00x0,0000
PWM6CR	PWM6 控制寄存器	FF64H	ENC6O	C6INI	-	C6_S[1:0]		EC6I	EC6T2SI	EC6T1SI	00x0,0000
PWM7CR	PWM7 控制寄存器	FF74H	ENC7O	C7INI	-	C7_S[1:0]		EC7I	EC7T2SI	EC7T1SI	00x0,0000
I2CMSCR	I^2C 主机控制寄存器	FE81H	EMSI	-	-	-	-	MSCMD[2:0]			0xxx,x000

I2CMSST	I^2C 主机状态寄存器	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I^2C 从机控制寄存器	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I^2C 从机状态寄存器	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000

11.4.1 中断使能寄存器（中断允许位）

IE（中断使能寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：总中断允许控制位。EA 的作用是使中断允许形成多级控制。即各中断源首先受 EA 控制;其次还受各中断源自己的中断允许控制位控制。

0：CPU 屏蔽所有的中断申请

1：CPU 开放中断

ELVD：低压检测中断允许位。

0：禁止低压检测中断

1：允许低压检测中断

EADC：A/D 转换中断允许位。

0：禁止 A/D 转换中断

1：允许 A/D 转换中断

ES：串行口 1 中断允许位。

0：禁止串行口 1 中断

1：允许串行口 1 中断

ET1：定时/计数器 T1 的溢出中断允许位。

0：禁止 T1 中断

1：允许 T1 中断

EX1：外部中断 1 中断允许位。

0：禁止 INT1 中断

1：允许 INT1 中断

ET0：定时/计数器 T0 的溢出中断允许位。

0：禁止 T0 中断

1：允许 T0 中断

EX0：外部中断 0 中断允许位。

0：禁止 INT0 中断

1：允许 INT0 中断

IE2（中断使能寄存器 2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ET4：定时/计数器 T4 的溢出中断允许位。

0：禁止 T4 中断

1：允许 T4 中断

ET3：定时/计数器 T3 的溢出中断允许位。

0：禁止 T3 中断

1：允许 T3 中断

STC MCU

ES4：串行口 4 中断允许位。

0：禁止串行口 4 中断

1：允许串行口 4 中断

ES3：串行口 3 中断允许位。

0：禁止串行口 3 中断

1：允许串行口 3 中断

ET2：定时/计数器 T2 的溢出中断允许位。

0：禁止 T2 中断

1：允许 T3 中断

ESPI：SPI 中断允许位。

0：禁止 SPI 中断

1：允许 SPI 中断

ES2：串行口 2 中断允许位。

0：禁止串行口 2 中断

1：允许串行口 2 中断

INTCLKO（外部中断与时钟输出控制寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

EX4：外部中断 4 中断允许位。

0：禁止 INT4 中断

1：允许 INT4 中断

EX3：外部中断 3 中断允许位。

0：禁止 INT3 中断

1：允许 INT3 中断

EX2：外部中断 2 中断允许位。

0：禁止 INT2 中断

1：允许 INT2 中断

PCA/CCP/PWM 中断控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	CIDL	-	-	-	CPS[2:0]			ECF
CCAPM0	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0
CCAPM1	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1
CCAPM2	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2
CCAPM3	DDH	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3

ECF：PCA 计数器中断允许位。

0：禁止 PCA 计数器中断

1：允许 PCA 计数器中断

ECCF0：PCA 模块 0 中断允许位。

0：禁止 PCA 模块 0 中断

1：允许 PCA 模块 0 中断

ECCF1：PCA 模块 1 中断允许位。

0：禁止 PCA 模块 1 中断

1：允许 PCA 模块 1 中断

ECCF2：PCA 模块 2 中断允许位。

0：禁止 PCA 模块 2 中断

1：允许 PCA 模块 2 中断

ECCF3：PCA 模块 3 中断允许位。

0：禁止 PCA 模块 3 中断

1：允许 PCA 模块 3 中断

CMPCR1（比较器控制寄存器 1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES

PIE：比较器上升沿中断允许位。

0：禁止比较器上升沿中断

1：允许比较器上升沿中断

NIE：比较器下降沿中断允许位。

0：禁止比较器下降沿中断

1：允许比较器下降沿中断

PWMCR（PWM 控制寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCR	FEH	ENPWM	ECBI	-	-	-	-	-	-

ECBI：增强PWM计数器中断允许位。

0：禁止 PWM 计数器中断

1：允许 PWM 计数器中断

PWMFDCR（PWM 异常检测控制寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMFDCR	F7H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF

EFDI：PWM外部异常事件中断允许位。

0：禁止 PWM 外部异常事件中断

1：允许 PWM 外部异常事件中断

增强型 PWM 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0CR	FF04H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI
PWM1CR	FF14H	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI
PWM2CR	FF24H	ENC2O	C2INI	-	C2_S[1:0]		EC2I	EC2T2SI	EC2T1SI
PWM3CR	FF34H	ENC3O	C3INI	-	C3_S[1:0]		EC3I	EC3T2SI	EC3T1SI
PWM4CR	FF44H	ENC4O	C4INI	-	C4_S[1:0]		EC4I	EC4T2SI	EC4T1SI
PWM5CR	FF54H	ENC5O	C5INI	-	C5_S[1:0]		EC5I	EC5T2SI	EC5T1SI
PWM6CR	FF64H	ENC6O	C6INI	-	C6_S[1:0]		EC6I	EC6T2SI	EC6T1SI
PWM7CR	FF74H	ENC7O	C7INI	-	C7_S[1:0]		EC7I	EC7T2SI	EC7T1SI

ECnI：PWM通道n中断允许位。

0：禁止第 n 通道 PWM 中断

1：允许第 n 通道 PWM 中断

ECnT2SI：PWM通道n第2个触发点中断允许位。

0：禁止第 n 通道 PWM 的第 2 个触发点中断

1：允许第 n 通道 PWM 的第 2 个触发点中断

ECnT1SI：PWM通道n第1个触发点中断允许位。

0：禁止第 n 通道 PWM 的第 1 个触发点中断

1：允许第 n 通道 PWM 的第 1 个触发点中断

I2C 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	FE81H	EMSI	-	-	-	-	MSCMD[2:0]		
I2CSLCR	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

EMSI：I^2C主机模式中断允许位。

0：禁止 I^2C 主机模式中断

1：允许 I^2C 主机模式中断

ESTAI：I^2C从机接收START事件中断允许位。

0：禁止 I^2C 从机接收 START 事件中断

1：允许 I^2C 从机接收 START 事件中断

ERXI：I^2C从机接收数据完成事件中断允许位。

0：禁止 I^2C 从机接收数据完成事件中断

1：允许 I^2C 从机接收数据完成事件中断

ETXI：I^2C从机发送数据完成事件中断允许位。

0：禁止 I^2C 从机发送数据完成事件中断

1：允许 I^2C 从机发送数据完成事件中断

ESTOI：I^2C从机接收STOP事件中断允许位。

0：禁止 I^2C 从机接收 STOP 事件中断

1：允许 I^2C 从机接收 STOP 事件中断

11.4.2 中断请求寄存器（中断标志位）

定时器控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1：定时器1溢出中断标志。中断服务程序中，硬件自动清零。

TF0：定时器0溢出中断标志。中断服务程序中，硬件自动清零。

IE1：外部中断1中断请求标志。中断服务程序中，硬件自动清零。

IE0：外部中断0中断请求标志。中断服务程序中，硬件自动清零。

中断标志辅助寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0

AUXINTIF	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF

INT4IF：外部中断4中断请求标志。需要软件清零。

INT3IF：外部中断3中断请求标志。需要软件清零。

INT2IF：外部中断2中断请求标志。需要软件清零。

T4IF：定时器4溢出中断标志。需要软件清零。

T3IF：定时器3溢出中断标志。需要软件清零。

T2IF：定时器2溢出中断标志。需要软件清零。

串口控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI
S2CON	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI
S3CON	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI
S4CON	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

TI：串口1发送完成中断请求标志。需要软件清零。

RI：串口1接收完成中断请求标志。需要软件清零。

S2TI：串口2发送完成中断请求标志。需要软件清零。

S2RI：串口2接收完成中断请求标志。需要软件清零。

S3TI：串口3发送完成中断请求标志。需要软件清零。

S3RI：串口3接收完成中断请求标志。需要软件清零。

S4TI：串口4发送完成中断请求标志。需要软件清零。

S4RI：串口4接收完成中断请求标志。需要软件清零。

电源管理寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

LVDF：低压检测中断请求标志。需要软件清零。

ADC 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]			

ADC_FLAG：ADC转换完成中断请求标志。需要软件清零。

SPI 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	SPIF	WCOL	-	-	-	-	-	-

SPIF：SPI数据传输完成中断请求标志。需要软件清零。

PCA 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCON	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0

CF：PCA计数器中断请求标志。需要软件清零。

CCF3：PCA模块3中断请求标志。需要软件清零。

CCF2：PCA模块2中断请求标志。需要软件清零。

CCF1：PCA模块1中断请求标志。需要软件清零。

CCF0：PCA模块0中断请求标志。需要软件清零。

比较器控制寄存器 1

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES

CMPIF：比较器中断请求标志。需要软件清零。

增强型 PWM 配置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCFG	F1H	CBIF	ETADC	-	-	-	-	-	-

CBIF：增强型PWM计数器中断请求标志。需要软件清零。

增强型 PWM 中断标志寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMIF	F6H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF

C7IF：增强型PWM通道7中断请求标志。需要软件清零。

C6IF：增强型PWM通道6中断请求标志。需要软件清零。

C5IF：增强型PWM通道5中断请求标志。需要软件清零。

C4IF：增强型PWM通道4中断请求标志。需要软件清零。

C3IF：增强型PWM通道3中断请求标志。需要软件清零。

C2IF：增强型PWM通道2中断请求标志。需要软件清零。

C1IF：增强型PWM通道1中断请求标志。需要软件清零。

C0IF：增强型PWM通道0中断请求标志。需要软件清零。

增强型 PWM 异常检测控制决寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMFDCR	F7H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF

FDIF：增强型PWM异常检测中断请求标志。需要软件清零。

I2C 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO
I2CSLST	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO

MSIF：I^2C主机模式中断请求标志。需要软件清零。

ESTAI：I^2C从机接收START事件中断请求标志。需要软件清零。

ERXI：I^2C从机接收数据完成事件中断请求标志。需要软件清零。

ETXI：I^2C从机发送数据完成事件中断请求标志。需要软件清零。

ESTOI：I^2C从机接收STOP事件中断请求标志。需要软件清零。

11.4.3 中断优先级寄存器

中断优先级控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0
IPH	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H
IP2	B5H		PI2C	PCMP	PX4	PPWMFD	PPWM	PSPI	PS2
IP2H	B6H		PI2CH	PCMPH	PX4H	PPWMFDH	PPWMH	PSPIH	PS2H

PX0H,PX0：外部中断0中断优先级控制位

00：INT0 中断优先级为 0 级（最低级）
01：INT0 中断优先级为 1 级（较低级）
10：INT0 中断优先级为 2 级（较高级）
11：INT0 中断优先级为 3 级（最高级）

PT0H,PT0：定时器0中断优先级控制位

00：定时器 0 中断优先级为 0 级（最低级）
01：定时器 0 中断优先级为 1 级（较低级）
10：定时器 0 中断优先级为 2 级（较高级）
11：定时器 0 中断优先级为 3 级（最高级）

PX1H,PX1：外部中断1中断优先级控制位

00：INT1 中断优先级为 0 级（最低级）
01：INT1 中断优先级为 1 级（较低级）
10：INT1 中断优先级为 2 级（较高级）
11：INT1 中断优先级为 3 级（最高级）

PT1H,PT1：定时器1中断优先级控制位

00：定时器 1 中断优先级为 0 级（最低级）
01：定时器 1 中断优先级为 1 级（较低级）
10：定时器 1 中断优先级为 2 级（较高级）
11：定时器 1 中断优先级为 3 级（最高级）

PSH,PS：串口1中断优先级控制位

00：串口 1 中断优先级为 0 级（最低级）
01：串口 1 中断优先级为 1 级（较低级）
10：串口 1 中断优先级为 2 级（较高级）
11：串口 1 中断优先级为 3 级（最高级）

PADCH,PADC：ADC中断优先级控制位

00：ADC 中断优先级为 0 级（最低级）
01：ADC 中断优先级为 1 级（较低级）
10：ADC 中断优先级为 2 级（较高级）
11：ADC 中断优先级为 3 级（最高级）

PLVDH,PLVD：低压检测中断优先级控制位

00：LVD 中断优先级为 0 级（最低级）
01：LVD 中断优先级为 1 级（较低级）

10：LVD 中断优先级为 2 级（较高级）
11：LVD 中断优先级为 3 级（最高级）

PPCAH,PPCA：CCP/PCA/PWM中断优先级控制位

00：CCP/PCA/PWM 中断优先级为 0 级（最低级）
01：CCP/PCA/PWM 中断优先级为 1 级（较低级）
10：CCP/PCA/PWM 中断优先级为 2 级（较高级）
11：CCP/PCA/PWM 中断优先级为 3 级（最高级）

PS2H,PS2：串口2中断优先级控制位

00：串口 2 中断优先级为 0 级（最低级）
01：串口 2 中断优先级为 1 级（较低级）
10：串口 2 中断优先级为 2 级（较高级）
11：串口 2 中断优先级为 3 级（最高级）

PSPIH,PSPI：SPI中断优先级控制位

00：SPI 中断优先级为 0 级（最低级）
01：SPI 中断优先级为 1 级（较低级）
10：SPI 中断优先级为 2 级（较高级）
11：SPI 中断优先级为 3 级（最高级）

PPWMH,PPWM：增强型PWM中断优先级控制位

00：增强型 PWM 中断优先级为 0 级（最低级）
01：增强型 PWM 中断优先级为 1 级（较低级）
10：增强型 PWM 中断优先级为 2 级（较高级）
11：增强型 PWM 中断优先级为 3 级（最高级）

PPWMFDH,PPWMFD：增强型PWM异常检测中断优先级控制位

00：PWMFD 中断优先级为 0 级（最低级）
01：PWMFD 中断优先级为 1 级（较低级）
10：PWMFD 中断优先级为 2 级（较高级）
11：PWMFD 中断优先级为 3 级（最高级）

PX4H,PX4：外部中断4中断优先级控制位

00：INT4 中断优先级为 0 级（最低级）
01：INT4 中断优先级为 1 级（较低级）
10：INT4 中断优先级为 2 级（较高级）
11：INT4 中断优先级为 3 级（最高级）

PCMPH,PCMP：比较器中断优先级控制位

00：CMP 中断优先级为 0 级（最低级）
01：CMP 中断优先级为 1 级（较低级）
10：CMP 中断优先级为 2 级（较高级）
11：CMP 中断优先级为 3 级（最高级）

PI2CH,PI2C：I2C中断优先级控制位

00：I2C 中断优先级为 0 级（最低级）
01：I2C 中断优先级为 1 级（较低级）
10：I2C 中断优先级为 2 级（较高级）
11：I2C 中断优先级为 3 级（最高级）

11.5 范例程序

11.5.1 INT0 中断（上升沿和下降沿）

汇编代码

```
                ORG     0000H
                LJMP    MAIN
                ORG     0003H
                LJMP    INT0ISR

                ORG     0100H
INT0ISR:
                JB      INT0,RISING             ;判断上升沿和下降沿
                CPL     P1.0                    ;测试端口
                RETI
RISING:
                CPL     P1.1                    ;测试端口
                RETI

MAIN:
                MOV     SP,#3FH

                CLR     IT0                     ;使能 INT0 上升沿和下降沿中断
                SETB    EX0                     ;使能 INT0 中断
                SETB    EA
                JMP     $

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

void INT0_Isr() interrupt 0
{
    if (INT0)                       //判断上升沿和下降沿
    {
        P10 = !P10;                 //测试端口
    }
    else
    {
        P11 = !P11;                 //测试端口
    }
}

void main()
{
    IT0 = 0;                        //使能 INT0 上升沿和下降沿中断
    EX0 = 1;                        //使能 INT0 中断
    EA = 1;

    while (1);
```

```
}
```

11.5.2 INT0 中断（下降沿）

汇编代码

```
                ORG     0000H
                LJMP    MAIN
                ORG     0003H
                LJMP    INT0ISR

                ORG     0100H
INT0ISR:
                CPL     P1.0                    ;测试端口
                RETI

MAIN:
                MOV     SP,#3FH

                SETB    IT0                     ;使能 INT0 下降沿中断
                SETB    EX0                     ;使能 INT0 中断
                SETB    EA
                JMP     $

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

sbit    P10     =   P1^0;

void INT0_Isr() interrupt 0
{
    P10 = !P10;                     //测试端口
}

void main()
{
    IT0 = 1;                        //使能 INT0 下降沿中断
    EX0 = 1;                        //使能 INT0 中断
    EA = 1;

    while (1);
}
```

11.5.3 INT1 中断（上升沿和下降沿）

汇编代码

```
                ORG     0000H
                LJMP    MAIN
                ORG     0013H
                LJMP    INT1ISR

                ORG     0100H
INT1ISR:
```

```
                JB      INT1,RISING             ;判断上升沿和下降沿
                CPL     P1.0                    ;测试端口
                RETI
RISING:
                CPL     P1.1                    ;测试端口
                RETI

MAIN:
                MOV     SP,#3FH

                CLR     IT1                     ;使能 INT1 上升沿和下降沿中断
                SETB    EX1                     ;使能 INT1 中断
                SETB    EA
                JMP     $

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

void INT1_Isr() interrupt 2
{
    if (INT1)                   //判断上升沿和下降沿
    {
        P10 = !P10;             //测试端口
    }
    else
    {
        P11 = !P11;             //测试端口
    }
}

void main()
{
    IT1 = 0;                    //使能 INT1 上升沿和下降沿中断
    EX1 = 1;                    //使能 INT1 中断
    EA = 1;

    while (1);
}
```

11.5.4 INT1 中断（下降沿）

汇编代码

```
                ORG     0000H
                LJMP    MAIN
                ORG     0013H
                LJMP    INT1ISR

                ORG     0100H
INT1ISR:
                CPL     P1.0                    ;测试端口
```

```
                RETI

MAIN:
                MOV         SP,#3FH

                SETB        IT1                 ;使能 INT1 下降沿中断
                SETB        EX1                 ;使能 INT1 中断
                SETB        EA
                JMP         $

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

sbit        P10             =   P1^0;

void INT1_Isr() interrupt 2
{
    P10 = !P10;                 //测试端口
}

void main()
{
    IT1 = 1;                    //使能 INT1 下降沿中断
    EX1 = 1;                    //使能 INT1 中断
    EA = 1;

    while (1);
}
```

11.5.5 INT2 中断（下降沿）

汇编代码

```
INTCLKO     DATA    8FH
EX2         EQU     10H
EX3         EQU     20H
EX4         EQU     40H

            ORG     0000H
            LJMP    MAIN
            ORG     0053H
            LJMP    INT2ISR

            ORG     0100H
INT2ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     INTCLKO,#EX2        ;使能 INT2 中断
            SETB    EA
            JMP     $
```

```
        END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

sfr     INTCLKO     =   0x8f;
#define EX2             0x10
#define EX3             0x20
#define EX4             0x40
sbit    P10         =   P1^0;

void INT2_Isr() interrupt 10
{
    P10 = !P10;                 //测试端口
}

void main()
{
    INTCLKO = EX2;              //使能 INT2 中断
    EA = 1;

    while (1);
}
```

11.5.6 INT3 中断（下降沿）

汇编代码

```
INTCLKO     DATA    8FH
EX2         EQU     10H
EX3         EQU     20H
EX4         EQU     40H

            ORG     0000H
            LJMP    MAIN
            ORG     005BH
            LJMP    INT3ISR

            ORG     0100H
INT3ISR:
            CPL     P1.0            ;测试端口
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     INTCLKO,#EX3    ;使能 INT3 中断
            SETB    EA
            JMP     $

            END
```

C 语言代码

```
#include "reg51.h"
```

```
#include "intrins.h"

sfr     INTCLKO     =   0x8f;
#define EX2             0x10
#define EX3             0x20
#define EX4             0x40
sbit    P10         =   P1^0;

void INT3_Isr() interrupt 11
{
    P10 = !P10;                 //测试端口
}

void main()
{
    INTCLKO = EX3;              //使能 INT3 中断
    EA = 1;

    while (1);
}
```

11.5.7 INT4 中断（下降沿）

汇编代码

```
INTCLKO     DATA    8FH
EX2         EQU     10H
EX3         EQU     20H
EX4         EQU     40H

            ORG     0000H
            LJMP    MAIN
            ORG     0083H
            LJMP    INT4ISR

            ORG     0100H
INT4ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     INTCLKO,#EX4        ;使能 INT4 中断
            SETB    EA
            JMP     $

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

sfr     INTCLKO     =   0x8f;
#define EX2             0x10
#define EX3             0x20
#define EX4             0x40
```

```
sbit        P10             =     P1^0;

void INT4_Isr() interrupt 16
{
    P10 = !P10;                     //测试端口
}

void main()
{
    INTCLKO = EX4;                  //使能 INT4 中断
    EA = 1;

    while (1);
}
```

11.5.8 定时器 0 中断

汇编代码

```
;测试工作频率为 11.0592MHz

            ORG         0000H
            LJMP        MAIN
            ORG         000BH
            LJMP        TM0ISR

            ORG         0100H
TM0ISR:
            CPL         P1.0                ;测试端口
            RETI

MAIN:
            MOV         SP,#3FH

            MOV         TMOD,#00H
            MOV         TL0,#66H            ;65536-11.0592M/12/1000
            MOV         TH0,#0FCH
            SETB        TR0                 ;启动定时器
            SETB        ET0                 ;使能定时器中断
            SETB        EA

            JMP         $

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为 11.0592MHz

sbit        P10             =     P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                     //测试端口
}
```

```
void main()
{
    TMOD = 0x00;
    TL0 = 0x66;                     //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                        //启动定时器
    ET0 = 1;                        //使能定时器中断
    EA = 1;

    while (1);
}
```

11.5.9 定时器 1 中断

汇编代码

```
;测试工作频率为11.0592MHz

            ORG     0000H
            LJMP    MAIN
            ORG     001BH
            LJMP    TM1ISR

            ORG     0100H
TM1ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     TMOD,#00H
            MOV     TL1,#66H            ;65536-11.0592M/12/1000
            MOV     TH1,#0FCH
            SETB    TR1                 ;启动定时器
            SETB    ET1                 ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为11.0592MHz

sbit    P10     =   P1^0;

void TM1_Isr() interrupt 3
{
    P10 = !P10;                     //测试端口
}

void main()
```

```
{
    TMOD = 0x00;
    TL1 = 0x66;                     //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                        //启动定时器
    ET1 = 1;                        //使能定时器中断
    EA = 1;

    while (1);
}
```

11.5.10 定时器 2 中断

汇编代码

```
;测试工作频率为11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H

            ORG     0000H
            LJMP    MAIN
            ORG     0063H
            LJMP    TM2ISR

            ORG     0100H
TM2ISR:
            CPL     P1.0            ;测试端口
            ANL     AUXINTIF,#NOT T2IF  ;清中断标志
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     T2L,#66H        ;65536-11.0592M/12/1000
            MOV     T2H,#0FCH
            MOV     AUXR,#10H       ;启动定时器
            MOV     IE2,#ET2        ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为11.0592MHz

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
```

```
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2         0x04
sfr     AUXINTIF=   0xef;
#define T2IF        0x01

sbit    P10     =   P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;                 //测试端口
    AUXINTIF &= ~T2IF;          //清中断标志
}

void main()
{
    T2L = 0x66;                 //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                //启动定时器
    IE2 = ET2;                  //使能定时器中断
    EA = 1;

    while (1);
}
```

11.5.11 定时器 3 中断

汇编代码

```
;测试工作频率为 11.0592MHz

T3L         DATA    0D5H
T3H         DATA    0D4H
T4T3M       DATA    0D1H
IE2         DATA    0AFH
ET3         EQU     20H
AUXINTIF    DATA    0EFH
T3IF        EQU     02H

            ORG     0000H
            LJMP    MAIN
            ORG     009BH
            LJMP    TM3ISR

            ORG     0100H
TM3ISR:
            CPL     P1.0                ;测试端口
            ANL     AUXINTIF,#NOT T3IF  ;清中断标志
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     T3L,#66H            ;65536-11.0592M/12/1000
            MOV     T3H,#0FCH
            MOV     T4T3M,#08H          ;启动定时器
            MOV     IE2,#ET3            ;使能定时器中断
            SETB    EA
```

```
        JMP     $

        END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为11.0592MHz

sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T4T3M   =   0xd1;
sfr     IE2     =   0xaf;
#define ET3             0x20
sfr     AUXINTIF=   0xef;
#define T3IF            0x02

sbit    P10     =   P1^0;

void TM3_Isr() interrupt 19
{
    P10 = !P10;                 //测试端口
    AUXINTIF &= ~T3IF;          //清中断标志
}

void main()
{
    T3L = 0x66;                 //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x08;               //启动定时器
    IE2 = ET3;                  //使能定时器中断
    EA = 1;

    while (1);
}
```

11.5.12 定时器 4 中断

汇编代码

```
;测试工作频率为11.0592MHz

T4L         DATA    0D3H
T4H         DATA    0D2H
T4T3M       DATA    0D1H
IE2         DATA    0AFH
ET4         EQU     40H
AUXINTIF    DATA    0EFH
T4IF        EQU     04H

            ORG     0000H
            LJMP    MAIN
            ORG     00A3H
            LJMP    TM4ISR

            ORG     0100H
```

```
TM4ISR:
        CPL     P1.0                ;测试端口
        ANL     AUXINTIF,#NOT T4IF  ;清中断标志
        RETI

MAIN:
        MOV     SP,#3FH

        MOV     T4L,#66H            ;65536-11.0592M/12/1000
        MOV     T4H,#0FCH
        MOV     T4T3M,#80H          ;启动定时器
        MOV     IE2,#ET4            ;使能定时器中断
        SETB    EA

        JMP     $

        END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为11.0592MHz

sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T4T3M   =   0xd1;
sfr     IE2     =   0xaf;
#define ET4         0x40
sfr     AUXINTIF=   0xef;
#define T4IF        0x04

sbit    P10     =   P1^0;

void TM4_Isr() interrupt 20
{
    P10 = !P10;                 //测试端口
    AUXINTIF &= ~T4IF;          //清中断标志
}

void main()
{
    T4L = 0x66;                 //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x80;               //启动定时器
    IE2 = ET4;                  //使能定时器中断
    EA = 1;

    while (1);
}
```

11.5.13 UART1 中断

汇编代码

```
;测试工作频率为11.0592MHz
```

```
T2L             DATA    0D7H
T2H             DATA    0D6H
AUXR            DATA    8EH

                ORG     0000H
                LJMP    MAIN
                ORG     0023H
                LJMP    UART1ISR

                ORG     0100H
UART1ISR:
                JNB     TI,CHECKRI
                CLR     TI                  ;清中断标志
                CPL     P1.0                ;测试端口
CHECKRI:
                JNB     RI,ISREXIT
                CLR     RI                  ;清中断标志
                CPL     P1.1                ;测试端口
ISREXIT:
                RETI

MAIN:
                MOV     SP,#3FH

                MOV     SCON,#50H
                MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
                MOV     T2H,#0FFH
                MOV     AUXR,#15H           ;启动定时器
                SETB    ES                  ;使能串口中断
                SETB    EA
                MOV     SBUF,#5AH           ;发送测试数据

                JMP     $

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为11.0592MHz

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

void UART1_Isr() interrupt 4
{
    if (TI)
    {
        TI = 0;                 //清中断标志
        P10 = !P10;             //测试端口
    }
    if (RI)
    {
```

```
            RI = 0;                     //清中断标志
            P11 = !P11;                 //测试端口
        }
}

void main()
{
        SCON = 0x50;
        T2L = 0xe8;                     //65536-11059200/115200/4=0FFE8H
        T2H = 0xff;
        AUXR = 0x15;                    //启动定时器
        ES = 1;                         //使能串口中断
        EA = 1;
        SBUF = 0x5a;                    //发送测试数据

        while (1);
}
```

11.5.14 UART2 中断

汇编代码

```
;测试工作频率为11.0592MHz

T2L             DATA    0D7H
T2H             DATA    0D6H
AUXR            DATA    8EH
S2CON           DATA    9AH
S2BUF           DATA    9BH
IE2             DATA    0AFH
ES2             EQU     01H

                ORG     0000H
                LJMP    MAIN
                ORG     0043H
                LJMP    UART2ISR

                ORG     0100H
UART2ISR:
                PUSH    ACC
                PUSH    PSW
                MOV     A,S2CON
                JNB     ACC.1,CHECKRI
                ANL     S2CON,#NOT 02H          ;清中断标志
                CPL     P1.2                    ;测试端口
CHECKRI:
                MOV     A,S2CON
                JNB     ACC.0,ISREXIT
                ANL     S2CON,#NOT 01H          ;清中断标志
                CPL     P1.3                    ;测试端口
ISREXIT:
                POP     PSW
                POP     ACC
                RETI

MAIN:
                MOV     SP,#3FH

                MOV     S2CON,#10H
```

```
        MOV     T2L,#0E8H               ;65536-11059200/115200/4=0FFE8H
        MOV     T2H,#0FFH
        MOV     AUXR,#14H               ;启动定时器
        MOV     IE2,#ES2                ;使能串口中断
        SETB    EA
        MOV     S2BUF,#5AH              ;发送测试数据

        JMP     $

        END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为11.0592MHz

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     S2CON   =   0x9a;
sfr     S2BUF   =   0x9b;
sfr     IE2     =   0xaf;
#define ES2         0x01

sbit    P12     =   P1^2;
sbit    P13     =   P1^3;

void UART2_Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;         //清中断标志
        P12 = !P12;             //测试端口
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;         //清中断标志
        P13 = !P13;             //测试端口
    }
}

void main()
{
    S2CON = 0x10;
    T2L = 0xe8;                 //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x14;                //启动定时器
    IE2 = ES2;                  //使能串口中断
    EA = 1;
    S2BUF = 0x5a;               //发送测试数据

    while (1);
}
```

STC MCU

11.5.15 UART3 中断

汇编代码

```
;测试工作频率为11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH
ES3         EQU     08H

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3ISR

            ORG     0100H
UART3ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     A,S3CON
            JNB     ACC.1,CHECKRI
            ANL     S3CON,#NOT 02H      ;清中断标志
            CPL     P1.0                ;测试端口
CHECKRI:
            MOV     A,S3CON
            JNB     ACC.0,ISREXIT
            ANL     S3CON,#NOT 01H      ;清中断标志
            CPL     P1.1                ;测试端口
ISREXIT:
            POP     PSW
            POP     ACC
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     S3CON,#10H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H           ;启动定时器
            MOV     IE2,#ES3            ;使能串口中断
            SETB    EA
            MOV     S3BUF,#5AH          ;发送测试数据

            JMP     $

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为11.0592MHz
```

```
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     S3CON   =   0xac;
sfr     S3BUF   =   0xad;
sfr     IE2     =   0xaf;
#define ES3         0x08

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

void UART3_Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;         //清中断标志
        P10 = !P10;             //测试端口
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;         //清中断标志
        P11 = !P11;             //测试端口
    }
}

void main()
{
    S3CON = 0x10;
    T2L = 0xe8;                 //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x14;                //启动定时器
    IE2 = ES3;                  //使能串口中断
    EA = 1;
    S3BUF = 0x5a;               //发送测试数据

    while (1);
}
```

11.5.16 UART4 中断

汇编代码

```
;测试工作频率为11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
S4CON       DATA    084H
S4BUF       DATA    085H
IE2         DATA    0AFH
ES4         EQU     10H

            ORG     0000H
            LJMP    MAIN
            ORG     0093H
            LJMP    UART4ISR

            ORG     0100H
UART4ISR:
```

```
                PUSH    ACC
                PUSH    PSW
                MOV     A,S4CON
                JNB     ACC.1,CHECKRI
                ANL     S4CON,#NOT 02H          ;清中断标志
                CPL     P1.0                    ;测试端口
CHECKRI:
                MOV     A,S4CON
                JNB     ACC.0,ISREXIT
                ANL     S4CON,#NOT 01H          ;清中断标志
                CPL     P1.1                    ;测试端口
ISREXIT:
                POP     PSW
                POP     ACC
                RETI

MAIN:
                MOV     SP,#3FH

                MOV     S4CON,#10H
                MOV     T2L,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     T2H,#0FFH
                MOV     AUXR,#14H               ;启动定时器
                MOV     IE2,#ES4                ;使能串口中断
                SETB    EA
                MOV     S4BUF,#5AH              ;发送测试数据

                JMP     $

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为 11.0592MHz

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     S4CON   =   0x84;
sfr     S4BUF   =   0x85;
sfr     IE2     =   0xaf;
#define ES4         0x10

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

void UART4_Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;             //清中断标志
        P10 = !P10;                 //测试端口
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;             //清中断标志
```

```
            P11 = !P11;                 //测试端口
        }
}

void main()
{
        S4CON = 0x10;
        T2L = 0xe8;                     //65536-11059200/115200/4=0FFE8H
        T2H = 0xff;
        AUXR = 0x14;                    //启动定时器
        IE2 = ES4;                      //使能串口中断
        EA = 1;
        S4BUF = 0x5a;                   //发送测试数据

        while (1);
}
```

11.5.17 ADC中断

汇编代码

```
ADC_CONTR   DATA    0BCH
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH
EADC        BIT     IE.5

            ORG     0000H
            LJMP    MAIN
            ORG     002BH
            LJMP    ADCISR

            ORG     0100H
ADCISR:
            ANL     ADC_CONTR,#NOT 20H      ;清中断标志
            MOV     P0,ADC_RES              ;测试端口
            MOV     P2,ADC_RESL             ;测试端口
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     ADCCFG,#00H
            MOV     ADC_CONTR,#0C0H         ;使能并启动 ADC 模块
            SETB    EADC                    ;使能 ADC 中断
            SETB    EA

            JMP     $

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

sfr     ADC_CONTR   =   0xbc;
sfr     ADC_RES     =   0xbd;
```

```
sfr     ADC_RESL    =   0xbe;
sfr     ADCCFG      =   0xde;
sbit    EADC        =   IE^5;

void ADC_Isr() interrupt 5
{
    ADC_CONTR &= ~0x20;             //清中断标志
    P0 = ADC_RES;                   //测试端口
    P2 = ADC_RESL;                  //测试端口
}

void main()
{
    ADCCFG = 0x00;
    ADC_CONTR = 0xc0;               //使能并启动 ADC 模块
    EADC = 1;                       //使能 ADC 中断
    EA = 1;

    while (1);
}
```

11.5.18 LVD中断

汇编代码

```
RSTCFG      DATA    0FFH
ENLVR       EQU     40H             ;RSTCFG.6
LVD2V2      EQU     00H             ;LVD@2.2V
LVD2V4      EQU     01H             ;LVD@2.4V
LVD2V7      EQU     02H             ;LVD@2.7V
LVD3V0      EQU     03H             ;LVD@3.0V
ELVD        BIT     IE.6
LVDF        EQU     20H             ;PCON.5

            ORG     0000H
            LJMP    MAIN
            ORG     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
            ANL     PCON,#NOT LVDF  ;清中断标志
            CPL     P1.0            ;测试端口
            RETI

MAIN:
            MOV     SP,#3FH

            ANL     PCON,#NOT LVDF  ;上电需要清中断标志
            MOV     RSTCFG,# LVD3V0 ;设置 LVD 电压为 3.0V
            SETB    ELVD            ;使能 LVD 中断
            SETB    EA
            JMP     $

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

sfr     RSTCFG      =   0xff;
#define ENLVR           0x40        //RSTCFG.6
#define LVD2V2          0x00        //LVD@2.2V
#define LVD2V4          0x01        //LVD@2.4V
#define LVD2V7          0x02        //LVD@2.7V
#define LVD3V0          0x03        //LVD@3.0V
sbit    ELVD        =   IE^6;
#define LVDF            0x20        //PCON.5
sbit    P10         =   P1^0;

void LVD_Isr() interrupt 6
{
    PCON &= ~LVDF;                  //清中断标志
    P10 = !P10;                     //测试端口
}

void main()
{
    PCON &= ~LVDF;                  //上电需要清中断标志
    RSTCFG = LVD3V0;                //设置 LVD 电压为 3.0V
    ELVD = 1;                       //使能 LVD 中断
    EA = 1;

    while (1);
}
```

11.5.19 PCA中断

汇编代码

```
;测试工作频率为 11.0592MHz

CCON        DATA    0D8H
CF          BIT     CCON.7
CR          BIT     CCON.6
CCF3        BIT     CCON.3
CCF2        BIT     CCON.2
CCF1        BIT     CCON.1
CCF0        BIT     CCON.0
CMOD        DATA    0D9H
CL          DATA    0E9H
CH          DATA    0F9H
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H
CCAPM3      DATA    0DDH
CCAP3L      DATA    0EDH
```

```
CCAP3H      DATA    0FDH
PCA_PWM3    DATA    0F5H

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            JNB     CF,ISREXIT
            CLR     CF              ;清中断标志
            CPL     P1.0            ;测试端口
ISREXIT:
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     CCON,#00H
            MOV     CMOD,#09H       ;PCA 时钟为系统时钟,使能 PCA 计时中断
            SETB    CR              ;启动 PCA 计时器
            SETB    EA

            JMP     $

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为 11.0592MHz

sfr     CCON        =   0xd8;
sbit    CF          =   CCON^7;
sbit    CR          =   CCON^6;
sbit    CCF3        =   CCON^3;
sbit    CCF2        =   CCON^2;
sbit    CCF1        =   CCON^1;
sbit    CCF0        =   CCON^0;
sfr     CMOD        =   0xd9;
sfr     CL          =   0xe9;
sfr     CH          =   0xf9;
sfr     CCAPM0      =   0xda;
sfr     CCAP0L      =   0xea;
sfr     CCAP0H      =   0xfa;
sfr     PCA_PWM0    =   0xf2;
sfr     CCAPM1      =   0xdb;
sfr     CCAP1L      =   0xeb;
sfr     CCAP1H      =   0xfb;
sfr     PCA_PWM1    =   0xf3;
sfr     CCAPM2      =   0xdc;
sfr     CCAP2L      =   0xec;
sfr     CCAP2H      =   0xfc;
sfr     PCA_PWM2    =   0xf4;
sfr     CCAPM3      =   0xdd;
sfr     CCAP3L      =   0xed;
```

STC MCU

```
sfr     CCAP3H      =   0xfd;
sfr     PCA_PWM3    =   0xf5;

sbit    P10         =   P1^0;

void PCA_Isr() interrupt 7
{
    if (CF)
    {
        CF = 0;                 //清中断标志
        P10 = !P10;             //测试端口
    }
}

void main()
{
    CCON = 0x00;
    CMOD = 0x09;                //PCA 时钟为系统时钟,使能 PCA 计时中断
    CR = 1;                     //启动 PCA 计时器
    EA = 1;

    while (1);
}
```

11.5.20 SPI中断

汇编代码

```
SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            MOV     SPSTAT,#0C0H        ;清中断标志
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     SPCTL,#50H          ;使能 SPI 主机模式
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     IE2,#ESPI           ;使能 SPI 中断
            SETB    EA
            MOV     SPDAT,#5AH          ;发送测试数据

            JMP     $

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

sfr     SPSTAT      =   0xcd;
sfr     SPCTL       =   0xce;
sfr     SPDAT       =   0xcf;
sfr     IE2         =   0xaf;
#define ESPI            0x02

sbit    P10         =   P1^0;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //清中断标志
    P10 = !P10;                     //测试端口
}

void main()
{
    SPCTL = 0x50;                   //使能 SPI 主机模式
    SPSTAT = 0xc0;                  //清中断标志
    IE2 = ESPI;                     //使能 SPI 中断
    EA = 1;
    SPDAT = 0x5a;                   //发送测试数据

    while (1);
}
```

11.5.21 CMP中断

汇编代码

```
CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

            ORG     0000H
            LJMP    MAIN
            ORG     00ABH
            LJMP    CMPISR

            ORG     0100H
CMPISR:
            ANL     CMPCR1,#NOT 40H     ;清中断标志
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     CMPCR2,#00H
            MOV     CMPCR1,#80H         ;使能比较器模块
            ORL     CMPCR1,#30H         ;使能比较器边沿中断
            ANL     CMPCR1,#NOT 08H     ;P3.6 为CMP+输入脚
            ORL     CMPCR1,#04H         ;P3.7 为CMP-输入脚
            ORL     CMPCR1,#02H         ;使能比较器输出
            SETB    EA
```

```
        JMP     $

        END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

sfr     CMPCR1  =   0xe6;
sfr     CMPCR2  =   0xe7;

sbit    P10     =   P1^0;

void CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;                //清中断标志
    P10 = !P10;                     //测试端口
}

void main()
{
    CMPCR2 = 0x00;
    CMPCR1 = 0x80;                  //使能比较器模块
    CMPCR1 |= 0x30;                 //使能比较器边沿中断
    CMPCR1 &= ~0x08;                //P3.6 为 CMP+输入脚
    CMPCR1 |= 0x04;                 //P3.7 为 CMP-输入脚
    CMPCR1 |= 0x02;                 //使能比较器输出
    EA = 1;

    while (1);
}
```

11.5.22 PWM中断

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

PWMCFG      DATA    0F1H
PWMIF       DATA    0F6H
PWMFDCR     DATA    0F7H
PWMCR       DATA    0FEH
PWMCH       XDATA   0FFF0H
PWMCL       XDATA   0FFF1H
PWMCKS      XDATA   0FFF2H
TADCPH      XDATA   0FFF3H
TADCPL      XDATA   0FFF4H
PWM0T1H     XDATA   0FF00H
PWM0T1L     XDATA   0FF01H
PWM0T2H     XDATA   0FF02H
PWM0T2L     XDATA   0FF03H
PWM0CR      XDATA   0FF04H
PWM0HLD     XDATA   0FF05H
PWM1T1H     XDATA   0FF10H
PWM1T1L     XDATA   0FF11H
```

```
PWM1T2H     XDATA   0FF12H
PWM1T2L     XDATA   0FF13H
PWM1CR      XDATA   0FF14H
PWM1HLD     XDATA   0FF15H
PWM2T1H     XDATA   0FF20H
PWM2T1L     XDATA   0FF21H
PWM2T2H     XDATA   0FF22H
PWM2T2L     XDATA   0FF23H
PWM2CR      XDATA   0FF24H
PWM2HLD     XDATA   0FF25H
PWM3T1H     XDATA   0FF30H
PWM3T1L     XDATA   0FF31H
PWM3T2H     XDATA   0FF32H
PWM3T2L     XDATA   0FF33H
PWM3CR      XDATA   0FF34H
PWM3HLD     XDATA   0FF35H
PWM4T1H     XDATA   0FF40H
PWM4T1L     XDATA   0FF41H
PWM4T2H     XDATA   0FF42H
PWM4T2L     XDATA   0FF43H
PWM4CR      XDATA   0FF44H
PWM4HLD     XDATA   0FF45H
PWM5T1H     XDATA   0FF50H
PWM5T1L     XDATA   0FF51H
PWM5T2H     XDATA   0FF52H
PWM5T2L     XDATA   0FF53H
PWM5CR      XDATA   0FF54H
PWM5HLD     XDATA   0FF55H
PWM6T1H     XDATA   0FF60H
PWM6T1L     XDATA   0FF61H
PWM6T2H     XDATA   0FF62H
PWM6T2L     XDATA   0FF63H
PWM6CR      XDATA   0FF64H
PWM6HLD     XDATA   0FF65H
PWM7T1H     XDATA   0FF70H
PWM7T1L     XDATA   0FF71H
PWM7T2H     XDATA   0FF72H
PWM7T2L     XDATA   0FF73H
PWM7CR      XDATA   0FF74H
PWM7HLD     XDATA   0FF75H

            ORG     0000H
            LJMP    MAIN
            ORG     00B3H
            LJMP    PWMISR
            ORG     00BBH
            LJMP    PWMFDISR

            ORG     0100H
PWMISR:
            PUSH    ACC
            PUSH    PSW
            MOV     A,PWMCFG
            JNB     ACC.7,ISREXIT
            ANL     PWMCFG,#NOT 80H     ;清中断标志
            CPL     P1.0                ;测试端口
ISREXIT:
            POP     PSW
            POP     ACC
```

STC MCU

```
                RETI

PWMFDISR:
                ANL     PWMFDCR,#NOT 01H        ;清中断标志
                CPL     P1.1                    ;测试端口
                RETI

MAIN:
                MOV     SP,#3FH

                MOV     P_SW2,#80H
                MOV     A,#0FH
                MOV     DPTR,#PWMCKS
                MOVX    @DPTR,A                 ;PWM 时钟为系统时钟/16
                MOV     A,#01H
                MOV     DPTR,#PWMCH             ;设置 PWM 周期为 256 个 PWM 时钟
                MOVX    @DPTR,A
                MOV     A,#00H
                MOV     DPTR,#PWMCL
                MOVX    @DPTR,A
                MOV     P_SW2,#00H

                MOV     PWMFDCR,#7AH            ;使能 IO 口异常检测中断
                MOV     PWMCR,#0C0H             ;启动 PWM 模块并使能 PWM 计数器中断
                SETB    EA

                JMP     $

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为 11.0592MHz

sfr     P_SW2       =   0xba;
sfr     PWMCFG      =   0xf1;
sfr     PWMIF       =   0xf6;
sfr     PWMFDCR     =   0xf7;
sfr     PWMCR       =   0xfe;

#define PWMC        (*(unsigned int volatile xdata *)0xfff0)
#define PWMCKS      (*(unsigned char volatile xdata *)0xfff2)
#define TADCP       (*(unsigned int volatile xdata *)0xfff3)
#define PWM0T1      (*(unsigned int volatile xdata *)0xff00)
#define PWM0T2      (*(unsigned int volatile xdata *)0xff02)
#define PWM0CR      (*(unsigned char volatile xdata *)0xff04)
#define PWM0HLD     (*(unsigned char volatile xdata *)0xff05)
#define PWM1T1      (*(unsigned int volatile xdata *)0xff10)
#define PWM1T2      (*(unsigned int volatile xdata *)0xff12)
#define PWM1CR      (*(unsigned char volatile xdata *)0xff14)
#define PWM1HLD     (*(unsigned char volatile xdata *)0xff15)
#define PWM2T1      (*(unsigned int volatile xdata *)0xff20)
#define PWM2T2      (*(unsigned int volatile xdata *)0xff22)
#define PWM2CR      (*(unsigned char volatile xdata *)0xff24)
#define PWM2HLD     (*(unsigned char volatile xdata *)0xff25)
#define PWM3T1      (*(unsigned int volatile xdata *)0xff30)
```

```
#define   PWM3T2        (*(unsigned int volatile xdata *)0xff32)
#define   PWM3CR        (*(unsigned char volatile xdata *)0xff34)
#define   PWM3HLD       (*(unsigned char volatile xdata *)0xff35)
#define   PWM4T1        (*(unsigned int volatile xdata *)0xff40)
#define   PWM4T2        (*(unsigned int volatile xdata *)0xff42)
#define   PWM4CR        (*(unsigned char volatile xdata *)0xff44)
#define   PWM4HLD       (*(unsigned char volatile xdata *)0xff45)
#define   PWM5T1        (*(unsigned int volatile xdata *)0xff50)
#define   PWM5T2        (*(unsigned int volatile xdata *)0xff52)
#define   PWM5CR        (*(unsigned char volatile xdata *)0xff54)
#define   PWM5HLD       (*(unsigned char volatile xdata *)0xff55)
#define   PWM6T1        (*(unsigned int volatile xdata *)0xff60)
#define   PWM6T2        (*(unsigned int volatile xdata *)0xff62)
#define   PWM6CR        (*(unsigned char volatile xdata *)0xff64)
#define   PWM6HLD       (*(unsigned char volatile xdata *)0xff65)
#define   PWM7T1        (*(unsigned int volatile xdata *)0xff70)
#define   PWM7T2        (*(unsigned int volatile xdata *)0xff72)
#define   PWM7CR        (*(unsigned char volatile xdata *)0xff74)
#define   PWM7HLD       (*(unsigned char volatile xdata *)0xff75)

sbit      P10           =   P1^0;
sbit      P11           =   P1^1;

void PWM_Isr() interrupt 22
{
    if (PWMCFG & 0x80)
    {
        PWMCFG &= ~0x80;            //清中断标志
        P10 = !P10;                 //测试端口
    }
}

void PWMFD_Isr() interrupt 23
{
    PWMFDCR &= ~0x01;               //清中断标志
    P11 = !P11;                     //测试端口
}

void main()
{
    P_SW2 = 0x80;
    PWMCKS = 0x0f;                  // PWM 时钟为系统时钟/16
    PWMC = 0x0100;                  //设置 PWM 周期为 256 个 PWM 时钟
    P_SW2 = 0x00;

    PWMFDCR = 0x7a;                 //能 IO 口异常检测中断
    PWMCR = 0xc0;                   //启动 PWM 模块并使能 PWM 计数器中断
    EA = 1;

    while (1);
}
```

11.5.23 I2C中断

汇编代码

```
P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
```

```
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

            ORG     0000H
            LJMP    MAIN
            ORG     00C3H
            LJMP    I2CISR

            ORG     0100H
I2CISR:
            PUSH    ACC
            PUSH    DPL
            PUSH    DPH
            PUSH    P_SW2
            MOV     P_SW2,#80H
            MOV     DPTR,#I2CMSST
            MOVX    A,@DPTR
            ANL     A,#NOT 40H          ;清中断标志
            MOVX    @DPTR,A
            CPL     P1.0                ;测试端口
            POP     P_SW2
            POP     DPH
            POP     DPL
            POP     ACC
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     P_SW2,#80H
            MOV     A,#0C0H             ;使能 I2C 主机模式
            MOV     DPTR,#I2CCFG
            MOVX    @DPTR,A
            MOV     A,#80H              ;使能 I2C 中断
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            MOV     P_SW2,#00H
            SETB    EA

            MOV     P_SW2,#80H
            MOV     A,#081H             ;发送起始命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            MOV     P_SW2,#00H

            JMP     $

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"
```

```
sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sbit    P10         =   P1^0;

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;           //清中断标志
        P10 = !P10;                 //测试端口
    }
    _pop_(P_SW2);
}

void main()
{
    P_SW2 = 0x80;
    I2CCFG = 0xc0;                  //使能 I2C 主机模式
    I2CMSCR = 0x80;                 //使能 I2C 中断;
    P_SW2 = 0x00;
    EA = 1;

    P_SW2 = 0x80;
    I2CMSCR = 0x81;                 //发送起始命令
    P_SW2 = 0x00;

    while (1);
}
```

STC MCU

12 定时器/计数器

特别注意：当需要使能定时器的时钟输出功能时，相关的时钟输出口可设置为准双向口模式或者推挽输出模式，但一定不能设置开漏模式，否则没有输出

STC8A/8F 系列单片机内部设置了 5 个 16 位定时器/计数器。5 个 16 位定时器 T0、T1、T2、T3 和 T4 都具有计数方式和定时方式两种工作方式。对定时器/计数器 T0 和 T1，用它们在特殊功能寄存器 TMOD 中相对应的控制位 C/T 来选择 T0 或 T1 为定时器还是计数器。对定时器/计数器 T2，用特殊功能寄存器 AUXR 中的控制位 T2_C/T 来选择 T2 为定时器还是计数器。对定时器/计数器 T3，用特殊功能寄存器 T4T3M 中的控制位 T3_C/T 来选择 T3 为定时器还是计数器。对定时器/计数器 T4，用特殊功能寄存器 T4T3M 中的控制位 T4_C/T 来选择 T4 为定时器还是计数器。定时器/计数器的核心部件是一个加法计数器，其本质是对脉冲进行计数。只是计数脉冲来源不同：如果计数脉冲来自系统时钟，则为定时方式，此时定时器/计数器每 12 个时钟或者每 1 个时钟得到一个计数脉冲，计数值加 1；如果计数脉冲来自单片机外部引脚，则为计数方式，每来一个脉冲加 1。

当定时器/计数器 T0、T1 及 T2 工作在定时模式时，特殊功能寄存器 AUXR 中的 T0x12、T1x12 和 T2x12 分别决定是系统时钟/12 还是系统时钟/1（不分频）后让 T0、T1 和 T2 进行计数。当定时器/计数器 T3 和 T4 工作在定时模式时，特殊功能寄存器 T4T3M 中的 T3x12 和 T4x12 分别决定是系统时钟/12 还是系统时钟/1（不分频）后让 T3 和 T4 进行计数。当定时器/计数器工作在计数模式时，对外部脉冲计数不分频。

定时器/计数器 0 有 4 种工作模式：模式 0（16 位自动重装载模式），模式 1（16 位不可重装载模式），模式 2（8 位自动重装模式），模式 3（不可屏蔽中断的 16 位自动重装载模式）。定时器/计数器 1 除模式 3 外，其他工作模式与定时器/计数器 0 相同。T1 在模式 3 时无效，停止计数。定时器 T2 的工作模式固定为 16 位自动重装载模式。T2 可以当定时器使用，也可以当串口的波特率发生器和可编程时钟输出。定时器 3、定时器 4 与定时器 T2 一样，它们的工作模式固定为 16 位自动重装载模式。T3/T4 可以当定时器使用，也可以当串口的波特率发生器和可编程时钟输出。

12.1 定时器的相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	定时器模式寄存器	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	定时器 0 低 8 位寄存器	8AH									0000,0000
TL1	定时器 1 低 8 位寄存器	8BH									0000,0000
TH0	定时器 0 高 8 位寄存器	8CH									0000,0000
TH1	定时器 1 高 8 位寄存器	8DH									0000,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
INTCLKO	中断与时钟输出控制寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
WKTCL	掉电唤醒定时器低字节	AAH									1111,1111
WKTCH	掉电唤醒定时器高字节	ABH	WKTEN								0111,1111
T4T3M	定时器 4/3 控制寄存器	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000
T4H	定时器 4 高字节	D2H									0000,0000
T4L	定时器 4 低字节	D3H									0000,0000
T3H	定时器 3 高字节	D4H									0000,0000

T3L	定时器 3 低字节	D5H		0000,0000
T2H	定时器 2 高字节	D6H		0000,0000
T2L	定时器 2 低字节	D7H		0000,0000

STC MCU

12.2 定时器 0/1

12.2.1 定时器 0/1 控制寄存器（TCON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1：T1溢出中断标志。T1被允许计数以后，从初值开始加1计数。当产生溢出时由硬件将TF1位置“1”，并向CPU请求中断，一直保持到CPU响应中断时，才由硬件清“0”（也可由查询软件清“0”）。

TR1：定时器T1的运行控制位。该位由软件置位和清零。当GATE（TMOD.7）=0，TR1=1时就允许T1开始计数，TR1=0时禁止T1计数。当GATE（TMOD.7）=1，TR1=1且INT1输入高电平时，才允许T1计数。

TF0：T0溢出中断标志。T0被允许计数以后，从初值开始加1计数，当产生溢出时，由硬件置“1”TF0，向CPU请求中断，一直保持CPU响应该中断时，才由硬件清0（也可由查询软件清0）。

TR0：定时器T0的运行控制位。该位由软件置位和清零。当GATE（TMOD.3）=0，TR0=1时就允许T0开始计数，TR0=0时禁止T0计数。当GATE（TMOD.3）=1，TR0=1且INT0输入高电平时，才允许T0计数，TR0=0时禁止T0计数。

IE1：外部中断1请求源（INT1/P3.3）标志。IE1=1，外部中断向CPU请求中断，当CPU响应该中断时由硬件清“0”IE1。

IT1：外部中断源1触发控制位。IT1=0，上升沿或下降沿均可触发外部中断1。IT1=1，外部中断1程控为下降沿触发方式。

IE0：外部中断0请求源（INT0/P3.2）标志。IE0=1外部中断0向CPU请求中断，当CPU响应外部中断时，由硬件清“0”IE0（边沿触发方式）。

IT0：外部中断源0触发控制位。IT0=0，上升沿或下降沿均可触发外部中断0。IT0=1，外部中断0程控为下降沿触发方式。

12.2.2 定时器 0/1 模式寄存器（TMOD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TMOD	89H	T1_GATE	T1_C/T	T1_M1	T1_M0	T0_GATE	T0_C/T	T0_M1	T0_M0

T1_GATE：控制定时器1，置1时只有在INT1脚为高及TR1控制位置1时才可打开定时器/计数器1。

T0_GATE：控制定时器0，置1时只有在INT0脚为高及TR0控制位置1时才可打开定时器/计数器0。

T1_C/T：控制定时器1用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T1/P3.5外部脉冲进行计数）。

T0_C/T：控制定时器0用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T0/P3.4外部脉冲进行计数）。

T1_M1/T1_M0：定时器定时器/计数器1模式选择

T1_M1	T1_M0	定时器/计数器1工作模式
0	0	16位自动重载模式 当[TH1,TL1]中的16位计数值溢出时，系统会自动将内部16位

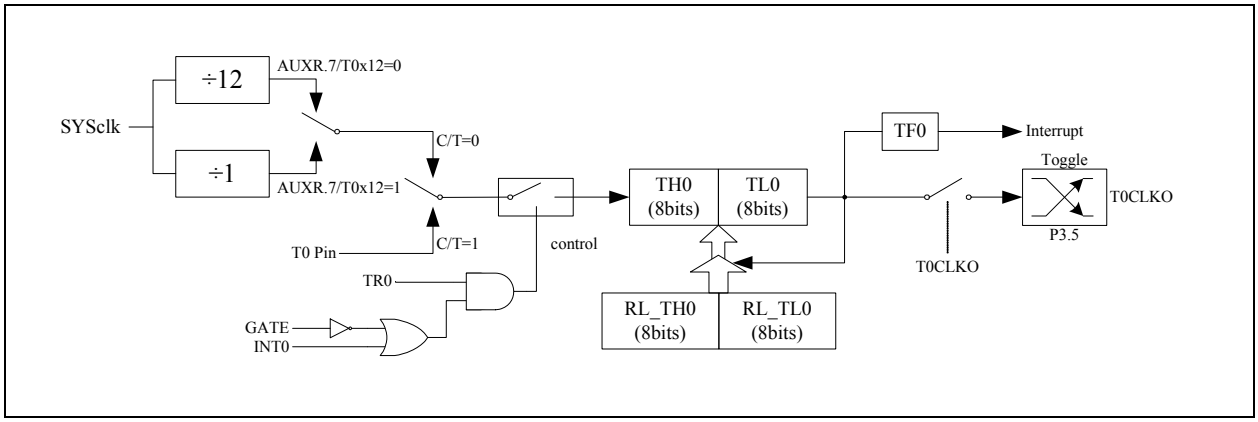
		重载寄存器中的重载值装入[TH1,TL1]中。
0	1	16位不自动重载模式 当[TH1,TL1]中的16位计数值溢出时，定时器1将从0开始计数
1	0	8位自动重载模式 当TL1中的8位计数值溢出时，系统会自动将TH1中的重载值装入TL1中。
1	1	T1停止工作

T0_M1/T0_M0：定时器定时器/计数器0模式选择

T0_M1	T0_M0	定时器/计数器0工作模式
0	0	16位自动重载模式 当[TH0,TL0]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[TH0,TL0]中。
0	1	16位不自动重载模式 当[TH0,TL0]中的16位计数值溢出时，定时器0将从0开始计数
1	0	8位自动重载模式 当TL0中的8位计数值溢出时，系统会自动将TH0中的重载值装入TL0中。
1	1	不可屏蔽中断的16位自动重载模式 与模式0相同，不可屏蔽中断，中断优先级最高，高于其他所有中断的优先级，并且不可关闭，可用作操作系统的系统节拍定时器，或者系统监控定时器。

12.2.3 定时器 0 模式 0（16 位自动重装载模式）

此模式下定时器/计数器 0 作为可自动重装载的 16 位计数器，如下图所示：



定时器/计数器 0 的模式 0：16 位自动重装载模式

当 GATE=0（TMOD.3）时，如 TR0=1，则定时器计数。GATE=1 时，允许由外部输入 INT0 控制定时器 0，这样可实现脉宽测量。TR0 为 TCON 寄存器内的控制位，TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

当 C/T=0 时，多路开关连接到系统时钟的分频输出，T0 对内部系统时钟计数，T0 工作在定时方式。当

C/T=1 时，多路开关连接到外部脉冲输入 P3.4/T0，即 T0 工作在计数方式。

STC 单片机的定时器 0 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T0 的速率由特殊功能寄存器 AUXR 中的 T0x12 决定，如果 T0x12=0，T0 则工作在 12T 模式；如果 T0x12=1，T0 则工作在 1T 模式

定时器 0 有两个隐藏的寄存器 RL_TH0 和 RL_TL0。RL_TH0 与 TH0 共有同一个地址，RL_TL0 与 TL0 共有同一个地址。当 TR0=0 即定时器/计数器 0 被禁止工作时，对 TL0 写入的内容会同时写入 RL_TL0，对 TH0 写入的内容也会同时写入 RL_TH0。当 TR0=1 即定时器/计数器 0 被允许工作时，对 TL0 写入内容，实际上不是写入当前寄存器 TL0 中，而是写入隐藏的寄存器 RL_TL0 中，对 TH0 写入内容，实际上也不是写入当前寄存器 TH0 中，而是写入隐藏的寄存器 RL_TH0，这样可以巧妙地实现 16 位重装载定时器。当读 TH0 和 TL0 的内容时，所读的内容就是 TH0 和 TL0 的内容，而不是 RL_TH0 和 RL_TL0 的内容。

当定时器 0 工作在模式 0（TMOD[1:0]/[M1,M0]=00B）时，[TH0,TL0]的溢出不仅置位 TF0，而且会自动将[RL_TH0,RL_TL0]的内容重新装入[TH0,TL0]。

当 T0CLKO/INT_CLKO.0=1 时，P3.5/T1 管脚配置为定时器 0 的时钟输出 T0CLKO。输出时钟频率为 T0 溢出率/2。

如果 C/T=0，定时器/计数器 T0 对内部系统时钟计数，则：

T0 工作在 1T 模式（AUXR.7/T0x12=1）时的输出时钟频率 = (SYSclk)/(65536-[RL_TH0, RL_TL0])/2

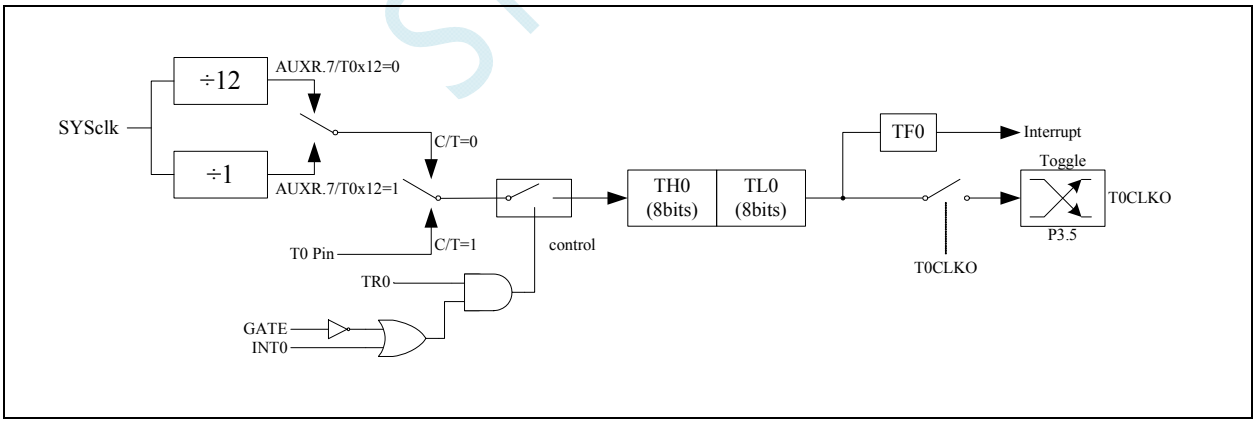
T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出时钟频率 = (SYSclk)/12/(65536-[RL_TH0, RL_TL0])/2

如果 C/T=1，定时器/计数器 T0 是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (65536-[RL_TH0, RL_TL0])/2

12.2.4 定时器 0 模式 1（16 位不可重装载模式）

此模式下定时器/计数器 0 工作在 16 位不可重装载模式，如下图所示



定时器/计数器 0 的模式 1：16 位不可重装载模式

此模式下，定时器/计数器 0 配置为 16 位不可重装载模式，由 TL0 的 8 位和 TH0 的 8 位所构成。TL0 的 8 位溢出向 TH0 进位，TH0 计数溢出置位 TCON 中的溢出标志位 TF0。

当 GATE=0(TMOD.3)时，如 TR0=1，则定时器计数。GATE=1 时，允许由外部输入 INT0 控制定时器 0，这样可实现脉宽测量。TR0 为 TCON 寄存器内的控制位，TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

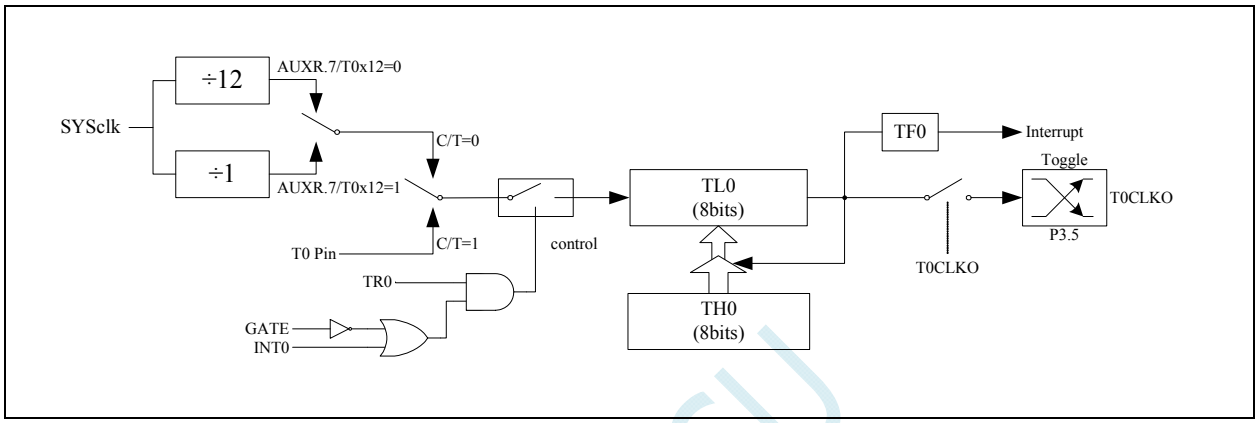
当 C/T=0 时，多路开关连接到系统时钟的分频输出，T0 对内部系统时钟计数，T0 工作在定时方式。当

C/T=1 时，多路开关连接到外部脉冲输入 P3.4/T0，即 T0 工作在计数方式。

STC 单片机的定时器 0 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T0 的速率由特殊功能寄存器 AUXR 中的 T0x12 决定，如果 T0x12=0，T0 则工作在 12T 模式；如果 T0x12=1，T0 则工作在 1T 模式

12.2.5 定时器 0 模式 2（8 位自动重装载模式）

此模式下定时器/计数器 0 作为可自动重装载的 8 位计数器，如下图所示：



定时器/计数器 0 的模式 2：8 位自动重装载模式

TL0 的溢出不仅置位 TF0，而且将 TH0 的内容重新装入 TL0，TH0 内容由软件预置，重装时 TH0 内容不变。

当 T0CLKO/INT_CLKO.0=1 时，P3.5/T1 管脚配置为定时器 0 的时钟输出 T0CLKO。输出时钟频率为 T0 溢出率/2。

如果 C/T=0，定时器/计数器 T0 对内部系统时钟计数，则：

T0 工作在 1T 模式（AUXR.7/T0x12=1）时的输出时钟频率 = (SYSclk)/(256-TH0)/2

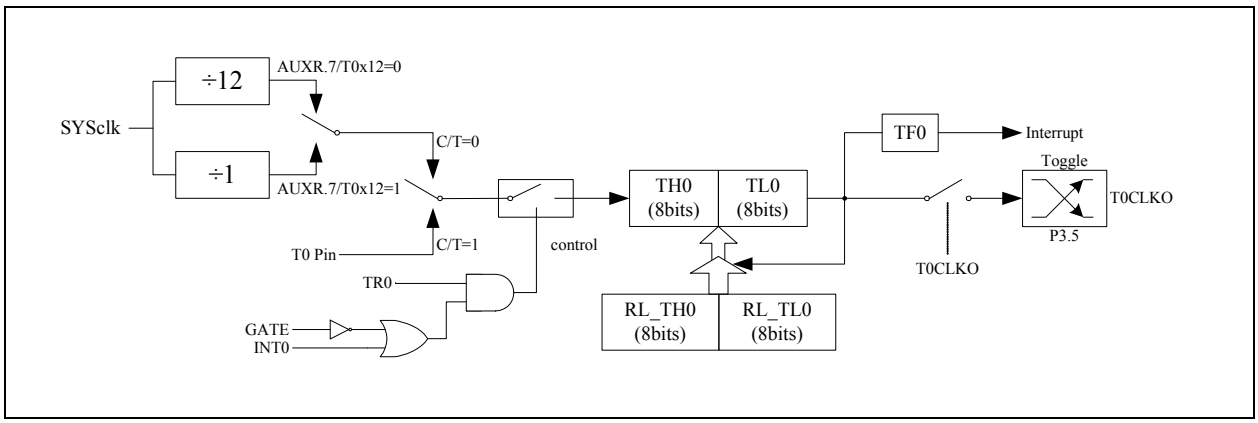
T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出时钟频率 = (SYSclk)/12/(256-TH0)/2

如果 C/T=1，定时器/计数器 T0 是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (256-TH0)/2

12.2.6 定时器 0 模式 3（不可屏蔽中断 16 位自动重装载，实时操作系统节拍器）

对定时器/计数器 0，其工作模式模式 3 与工作模式 0 是一样的（下图定时器模式 3 的原理图，与工作模式 0 是一样的）。唯一不同的是：当定时器/计数器 0 工作在模式 3 时，只需允许 ET0/IE.1(定时器/计数器 0 中断允许位)，不需要允许 EA/IE.7(总中断使能位)就能打开定时器/计数器 0 的中断，此模式下的定时器/计数器 0 中断与总中断使能位 EA 无关，一旦工作在模式 3 下的定时器/计数器 0 中断被打开(ET0=1)，那么该中断是不可屏蔽的，该中断的优先级是最高的，即该中断不能被任何中断所打断，而且该中断打开后既不受 EA/IE.7 控制也不再受 ET0 控制，当 EA=0 或 ET0=0 时都不能屏蔽此中断。故将此模式称为不可屏蔽中断的 16 位自动重装载模式。

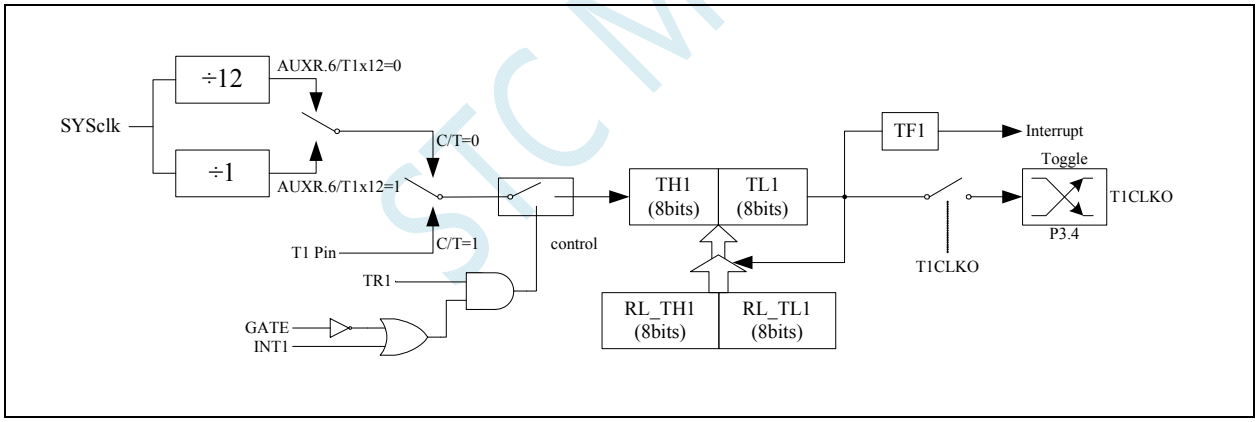


定时器/计数器 0 的模式 3：不可屏蔽中断的 16 位自动重装载模式

注意：当定时器/计数器 0 工作在模式 3(不可屏蔽中断的 16 位自动重装载模式)时，不需要允许 EA/IE.7(总中断使能位)，只需允许 ET0/IE.1(定时器/计数器 0 中断允许位)就能打开定时器/计数器 0 的中断，此模式下的定时器/计数器 0 中断与总中断使能位 EA 无关。一旦此模式下的定时器/计数器 0 中断被打开后，该定时器/计数器 0 中断优先级就是最高的，它不能被其它任何中断所打断(不管是比定时器/计数器 0 中断优先级低的中断还是比其优先级高的中断，都不能打断此时的定时器/计数器 0 中断)，而且该中断打开后既不受 EA/IE.7 控制也不再受 ET0 控制了，清零 EA 或 ET0 都不能关闭此中断。

12.2.7 定时器 1 模式 0（16 位自动重装载模式）

此模式下定时器/计数器 1 作为可自动重装载的 16 位计数器，如下图所示：



定时器/计数器 1 的模式 0：16 位自动重装载模式

当 GATE=0（TMOD.7）时，如 TR1=1，则定时器计数。GATE=1 时，允许由外部输入 INT1 控制定时器 1，这样可实现脉宽测量。TR1 为 TCON 寄存器内的控制位，TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

当 C/T=0 时，多路开关连接到系统时钟的分频输出，T1 对内部系统时钟计数，T1 工作在定时方式。当 C/T=1 时，多路开关连接到外部脉冲输入 P3.5/T1，即 T1 工作在计数方式。

STC 单片机的定时器 1 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T1 的速率由特殊功能寄存器 AUXR 中的 T1x12 决定，如果 T1x12=0，T1 则工作在 12T 模式；如果 T1x12=1，T1 则工作在 1T 模式

定时器 1 有两个隐藏的寄存器 RL_TH1 和 RL_TL1。RL_TH1 与 TH1 共有同一个地址，RL_TL1 与 TL1 共有同一个地址。当 TR1=0 即定时器/计数器 1 被禁止工作时，对 TL1 写入的内容会同时写入 RL_TL1，对

TH1 写入的内容也会同时写入 RL_TH1。当 TR1=1 即定时器/计数器 1 被允许工作时，对 TL1 写入内容，实际上不是写入当前寄存器 TL1 中，而是写入隐藏的寄存器 RL_TL1 中，对 TH1 写入内容，实际上也不是写入当前寄存器 TH1 中，而是写入隐藏的寄存器 RL_TH1，这样可以巧妙地实现 16 位重装载定时器。当读 TH1 和 TL1 的内容时，所读的内容就是 TH1 和 TL1 的内容，而不是 RL_TH1 和 RL_TL1 的内容。

当定时器 1 工作在模式 1（TMOD[5:4]/[M1,M0]=00B）时，[TH1,TL1]的溢出不仅置位 TF1，而且会自动将[RL_TH1,RL_TL1]的内容重新装入[TH1,TL1]。

当 T1CLKO/INT_CLKO.1=1 时，P3.4/T0 管脚配置为定时器 1 的时钟输出 T1CLKO。输出时钟频率为 T1 溢出率/2。

如果 C/T=0，定时器/计数器 T1 对内部系统时钟计数，则：

T1 工作在 1T 模式（AUXR.6/T1x12=1）时的输出时钟频率 = (SYSclk)/(65536-[RL_TH1, RL_TL1])/2

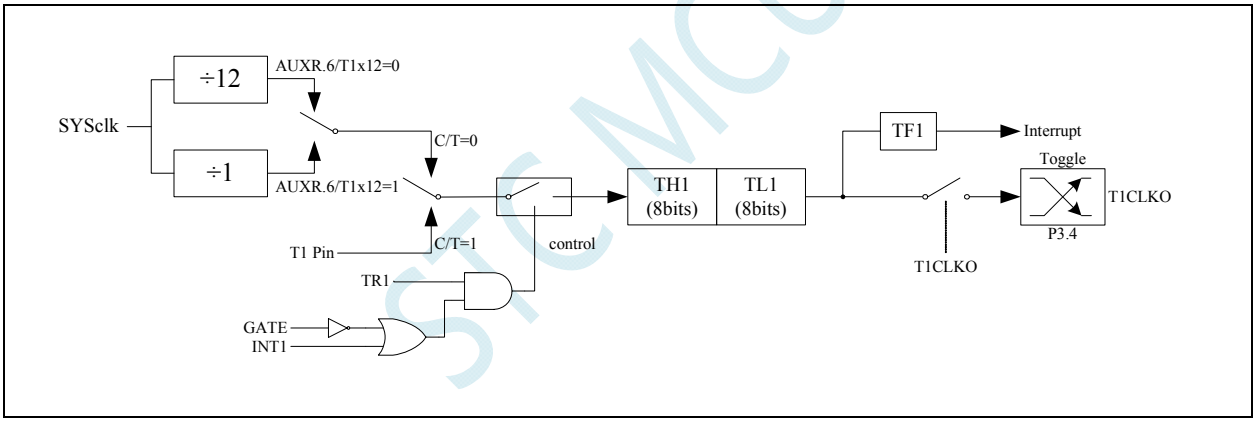
T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出时钟频率 = (SYSclk)/12/(65536-[RL_TH1, RL_TL1])/2

如果 C/T=1，定时器/计数器 T1 是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (65536-[RL_TH1, RL_TL1])/2

12.2.8 定时器 1 模式 1（16 位不可重装载模式）

此模式下定时器/计数器 1 工作在 16 位不可重装载模式，如下图所示



定时器/计数器 1 的模式 1：16 位不可重装载模式

此模式下，定时器/计数器 1 配置为 16 位不可重装载模式，由 TL1 的 8 位和 TH1 的 8 位所构成。TL1 的 8 位溢出向 TH1 进位，TH1 计数溢出置位 TCON 中的溢出标志位 TF1。

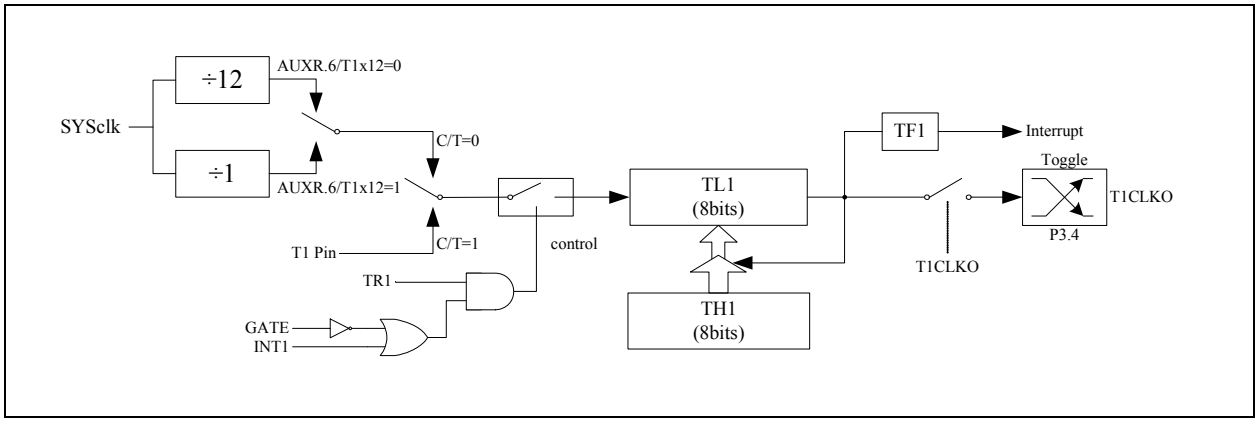
当 GATE=0(TMOD.7)时，如 TR1=1，则定时器计数。GATE=1 时，允许由外部输入 INT1 控制定时器 1，这样可实现脉宽测量。TR1 为 TCON 寄存器内的控制位，TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

当 C/T=0 时，多路开关连接到系统时钟的分频输出，T1 对内部系统时钟计数，T1 工作在定时方式。当 C/T=1 时，多路开关连接到外部脉冲输入 P3.5/T1，即 T1 工作在计数方式。

STC 单片机的定时器 1 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T1 的速率由特殊功能寄存器 AUXR 中的 T1x12 决定，如果 T1x12=0，T1 则工作在 12T 模式；如果 T1x12=1，T1 则工作在 1T 模式

12.2.9 定时器 1 模式 2（8 位自动重装载模式）

此模式下定时器/计数器 1 作为可自动重装载的 8 位计数器，如下图所示：



定时器/计数器 1 的模式 2：8 位自动重装载模式

TL1 的溢出不仅置位 TF1，而且将 TH1 的内容重新装入 TL1，TH1 内容由软件预置，重装时 TH1 内容不变。

当 T1CLKO/INT_CLKO.1=1 时，P3.4/T0 管脚配置为定时器 1 的时钟输出 T1CLKO。输出时钟频率为 T1 溢出率/2。

如果 C/T=0，定时器/计数器 T1 对内部系统时钟计数，则：

T1 工作在 1T 模式（AUXR.6/T1x12=1）时的输出时钟频率 = (SYSclk)/(256-TH1)/2

T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出时钟频率 = (SYSclk)/12/(256-TH1)/2

如果 C/T=1，定时器/计数器 T1 是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (256-TH1)/2

12.2.10 定时器 0 计数寄存器（TL0，TH0）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TL0	8AH								
TH0	8CH								

当定时器/计数器0工作在16位模式（模式0、模式1、模式3）时，TL0和TH0组合成为一个16位寄存器，TL0为低字节，TH0为高字节。若为8位模式（模式2）时，TL0和TH0为两个独立的8位寄存器。

12.2.11 定时器 1 计数寄存器（TL1，TH1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TL1	8BH								
TH1	8DH								

当定时器/计数器1工作在16位模式（模式0、模式1）时，TL1和TH1组合成为一个16位寄存器，TL1为低字节，TH1为高字节。若为8位模式（模式2）时，TL1和TH1为两个独立的8位寄存器。

12.2.12 辅助寄存器 1（AUXR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0

AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

T0x12：定时器0速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

T1x12：定时器1速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

12.2.13 中断与时钟输出控制寄存器（INTCLKO）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

T0CLKO：定时器0时钟输出控制

0：关闭时钟输出

1：使能 P3.5 口的是定时器 0 时钟输出功能

当定时器 0 计数发生溢出时，P3.5 口的电平自动发生翻转。

T1CLKO：定时器1时钟输出控制

0：关闭时钟输出

1：使能 P3.4 口的是定时器 1 时钟输出功能

当定时器 1 计数发生溢出时，P3.4 口的电平自动发生翻转。

12.2.14 定时器 0 定时计算公式

定时器模式	定时器速度	周期计算公式
模式0/3 （16位自动重载）	1T	$\text{定时器周期} = \frac{65536 - [TH0, TL0]}{SYSclk}$ （自动重载）
	12T	$\text{定时器周期} = \frac{65536 - [TH0, TL0]}{SYSclk} \times 12$ （自动重载）
模式1 （16位不自动重载）	1T	$\text{定时器周期} = \frac{65536 - [TH0, TL0]}{SYSclk}$ （需软件装载）
	12T	$\text{定时器周期} = \frac{65536 - [TH0, TL0]}{SYSclk} \times 12$ （需软件装载）
模式2 （8位自动重载）	1T	$\text{定时器周期} = \frac{256 - TH0}{SYSclk}$ （自动重载）
	12T	$\text{定时器周期} = \frac{256 - TH0}{SYSclk} \times 12$ （自动重载）

STC MCU

12.2.15 定时器 1 定时计算公式

定时器模式	定时器速度	周期计算公式
模式0 （16位自动重载）	1T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk}$ （自动重载）
	12T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk} \times 12$ （自动重载）
模式1 （16位不自动重载）	1T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk}$ （需软件装载）
	12T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk} \times 12$ （需软件装载）
模式2 （8位自动重载）	1T	定时器周期 = $\frac{256 - TH1}{SYSclk}$ （自动重载）
	12T	定时器周期 = $\frac{256 - TH1}{SYSclk} \times 12$ （自动重载）

STC MCU

12.3 定时器 2

12.3.1 辅助寄存器 1（AUXR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

T2R：定时器2的运行控制位

0：定时器 2 停止计数

1：定时器 2 开始计数

T2_C/T：控制定时器0用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T2/P1.2外部脉冲进行计数）。

T2x12：定时器2速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

12.3.2 中断与时钟输出控制寄存器（INTCLKO）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

T2CLKO：定时器2时钟输出控制

0：关闭时钟输出

1：使能 P1.3 口的是定时器 2 时钟输出功能

当定时器 2 计数发生溢出时，P1.3 口的电平自动发生翻转。

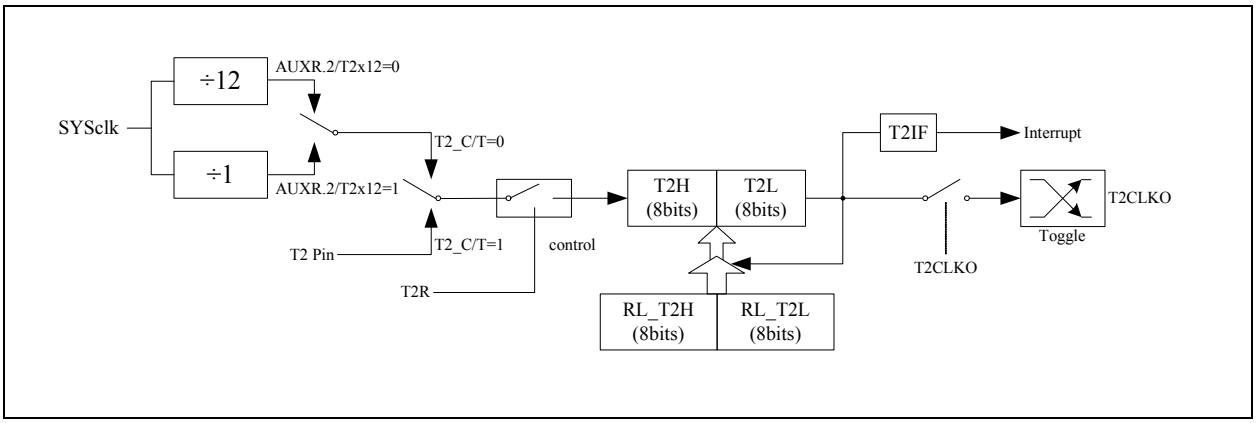
12.3.3 定时器 2 计数寄存器（T2L，T2H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T2L	D7H								
T2H	D6H								

定时器/计数器2的工作模式固定为16位重载模式，T2L和T2H组合成为一个16位寄存器，T2L为低字节，T2H为高字节。当[T2H,T2L]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[T2H,T2L]中。

12.3.4 定时器 2 工作模式

定时器/计数器 2 的原理框图如下：



定时器/计数器 2 的工作模式：16 位自动重装载模式

T2R/AUXR.4 为 AUXR 寄存器内的控制位，AUXR 寄存器各位的具体功能描述见上节 AUXR 寄存器的介绍。

当 T2_C/T=0 时，多路开关连接到系统时钟输出，T2 对内部系统时钟计数，T2 工作在定时方式。当 T2_C/T=1 时，多路开关连接到外部脉冲输 T2，即 T2 工作在计数方式。

STC 单片机的定时器 2 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T2 的速率由特殊功能寄存器 AUXR 中的 T2x12 决定，如果 T2x12=0，T2 则工作在 12T 模式；如果 T2x12=1，T2 则工作在 1T 模式

定时器 2 有两个隐藏的寄存器 RL_T2H 和 RL_T2L。RL_T2H 与 T2H 共有同一个地址，RL_T2L 与 T2L 共有同一个地址。当 T2R=0 即定时器/计数器 2 被禁止工作时，对 T2L 写入的内容会同时写入 RL_T2L，对 T2H 写入的内容也会同时写入 RL_T2H。当 T2R=1 即定时器/计数器 2 被允许工作时，对 T2L 写入内容，实际上不是写入当前寄存器 T2L 中，而是写入隐藏的寄存器 RL_T2L 中，对 T2H 写入内容，实际上也不是写入当前寄存器 T2H 中，而是写入隐藏的寄存器 RL_T2H，这样可以巧妙地实现 16 位重装载定时器。当读 T2H 和 T2L 的内容时，所读的内容就是 T2H 和 T2L 的内容，而不是 RL_T2H 和 RL_T2L 的内容。

[T2H,T2L]的溢出不仅置位中断请求标志位（T2IF），使 CPU 转去执行定时器 2 的中断程序，而且会自动将[RL_T2H,RL_T2L]的内容重新装入[T2H,T2L]。

12.3.5 定时器 2 计算公式

定时器速度	周期计算公式
1T	定时器周期 $= \frac{65536 - [T2H, T2L]}{SYSclk}$ （自动重载）
12T	定时器周期 $= \frac{65536 - [T2H, T2L]}{SYSclk} \times 12$ （自动重载）

12.4 定时器 3/4

12.4.1 定时器 4/3 控制寄存器（T4T3M）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

T4R：定时器4的运行控制位

0：定时器 4 停止计数

1：定时器 4 开始计数

T4_C/T：控制定时器4用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T4/P0.6外部脉冲进行计数）。

T4x12：定时器4速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

T4CLKO：定时器4时钟输出控制

0：关闭时钟输出

1：使能 P0.7 口的是定时器 4 时钟输出功能

当定时器 4 计数发生溢出时，P0.7 口的电平自动发生翻转。

T3R：定时器3的运行控制位

0：定时器 3 停止计数

1：定时器 3 开始计数

T3_C/T：控制定时器3用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T3/P0.4外部脉冲进行计数）。

T3x12：定时器3速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

T3CLKO：定时器3时钟输出控制

0：关闭时钟输出

1：使能 P0.5 口的是定时器 3 时钟输出功能

当定时器 3 计数发生溢出时，P0.5 口的电平自动发生翻转。

12.4.2 定时器 3 计数寄存器（T3L，T3H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T3L	D5H								
T3H	D4H								

定时器/计数器3的工作模式固定为16位重载模式，T3L和T3H组合成为一个16位寄存器，T3L为低字节，T3H为高字节。当[T3H,T3L]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[T3H,T3L]中。

12.4.3 定时器 4 计数寄存器（T4L，T4H）

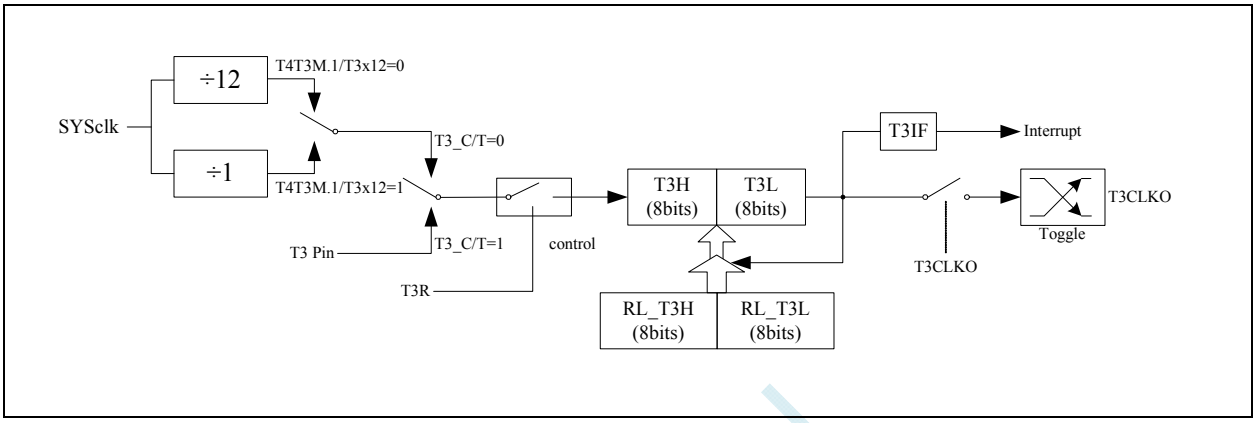
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T4L	D3H								

T4H	D2H	

定时器/计数器 4 的工作模式固定为 16 位重载模式，T4L 和 T4H 组合成为一个 16 位寄存器，T4L 为低字节，T4H 为高字节。当[T4H,T4L]中的 16 位计数值溢出时，系统会自动将内部 16 位重载寄存器中的重载值装入[T4H,T4L]中。

12.4.4 定时器 3 工作模式

定时器/计数器 3 的原理框图如下：



定时器/计数器 3 的工作模式：16 位自动重装载模式

T3R/T4T3M.3 为 T4T3M 寄存器内的控制位，T4T3M 寄存器各位的具体功能描述见上节 T4T3M 寄存器的介绍。

当 T3_C/T=0 时，多路开关连接到系统时钟输出，T3 对内部系统时钟计数，T3 工作在定时方式。当 T3_C/T=1 时，多路开关连接到外部脉冲输 T3，即 T3 工作在计数方式。

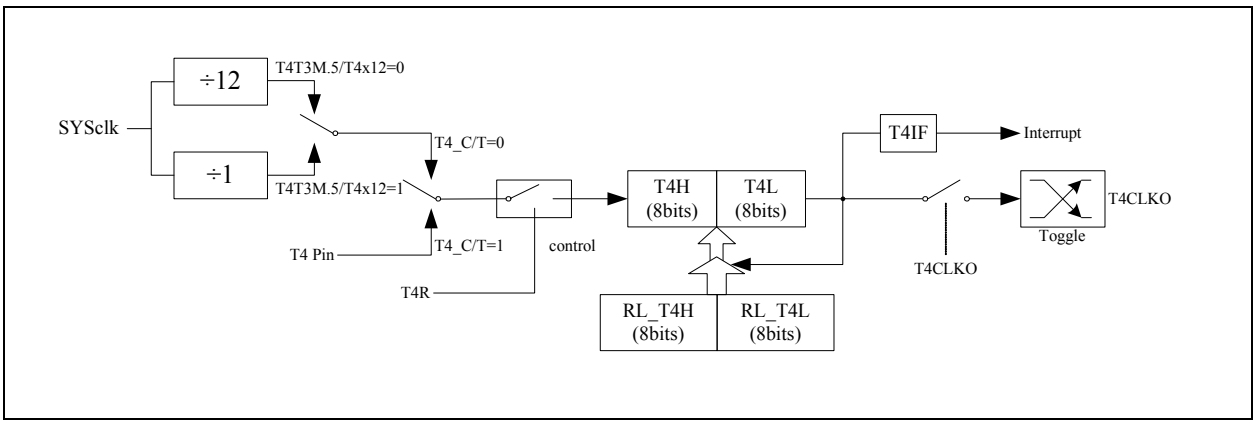
STC 单片机的定时器 3 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T3 的速率由特殊功能寄存器 T4T3M 中的 T3x12 决定，如果 T3x12=0，T3 则工作在 12T 模式；如果 T3x12=1，T3 则工作在 1T 模式

定时器 3 有两个隐藏的寄存器 RL_T3H 和 RL_T3L。RL_T3H 与 T3H 共有同一个地址，RL_T3L 与 T3L 共有同一个地址。当 T3R=0 即定时器/计数器 3 被禁止工作时，对 T3L 写入的内容会同时写入 RL_T3L，对 T3H 写入的内容也会同时写入 RL_T3H。当 T3R=1 即定时器/计数器 3 被允许工作时，对 T3L 写入内容，实际上不是写入当前寄存器 T3L 中，而是写入隐藏的寄存器 RL_T3L 中，对 T3H 写入内容，实际上也不是写入当前寄存器 T3H 中，而是写入隐藏的寄存器 RL_T3H，这样可以巧妙地实现 16 位重装载定时器。当读 T3H 和 T3L 的内容时，所读的内容就是 T3H 和 T3L 的内容，而不是 RL_T3H 和 RL_T3L 的内容。

[T3H,T3L]的溢出不仅置位中断请求标志位（T3IF），使 CPU 转去执行定时器 3 的中断程序，而且会自动将[RL_T3H,RL_T3L]的内容重新装入[T3H,T3L]。

12.4.5 定时器 4 工作模式

定时器/计数器 4 的原理框图如下：



定时器/计数器 4 的工作模式：16 位自动重装载模式

T4R/T4T3M.7 为 T4T3M 寄存器内的控制位，T4T3M 寄存器各位的具体功能描述见上节 T4T3M 寄存器的介绍。

当 T4_C/T=0 时，多路开关连接到系统时钟输出，T4 对内部系统时钟计数，T4 工作在定时方式。当 T4_C/T=1 时，多路开关连接到外部脉冲输 T4，即 T4 工作在计数方式。

STC 单片机的定时器 4 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T4 的速率由特殊功能寄存器 T4T3M 中的 T4x12 决定，如果 T4x12=0，T4 则工作在 12T 模式；如果 T4x12=1，T4 则工作在 1T 模式

定时器 4 有两个隐藏的寄存器 RL_T4H 和 RL_T4L。RL_T4H 与 T4H 共有同一个地址，RL_T4L 与 T4L 共有同一个地址。当 T4R=0 即定时器/计数器 4 被禁止工作时，对 T4L 写入的内容会同时写入 RL_T4L，对 T4H 写入的内容也会同时写入 RL_T4H。当 T4R=1 即定时器/计数器 4 被允许工作时，对 T4L 写入内容，实际上不是写入当前寄存器 T4L 中，而是写入隐藏的寄存器 RL_T4L 中，对 T4H 写入内容，实际上也不是写入当前寄存器 T4H 中，而是写入隐藏的寄存器 RL_T4H，这样可以巧妙地实现 16 位重装载定时器。当读 T4H 和 T4L 的内容时，所读的内容就是 T4H 和 T4L 的内容，而不是 RL_T4H 和 RL_T4L 的内容。

[T4H,T4L]的溢出不仅置位中断请求标志位（T4IF），使 CPU 转去执行定时器 4 的中断程序，而且会自动将[RL_T4H,RL_T4L]的内容重新装入[T4H,T4L]。

12.4.6 定时器 3 计算公式

定时器速度	周期计算公式
1T	定时器周期 $= \frac{65536 - [T3H, T3L]}{SYSclk}$ （自动重载）
12T	定时器周期 $= \frac{65536 - [T3H, T3L]}{SYSclk} \times 12$ （自动重载）

12.4.7 定时器 4 计算公式

定时器速度	周期计算公式
1T	定时器周期 = $\frac{65536 - [T4H, T4L]}{SYSclk}$ （自动重载）
12T	定时器周期 = $\frac{65536 - [T4H, T4L]}{SYSclk} \times 12$ （自动重载）

STC MCU

12.5 范例程序

12.5.1 定时器 0（模式 0－16 位自动重载），用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        P10         =   P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;                                //模式 0
    TL0 = 0x66;                                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                    //启动定时器
    ET0 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

STC MCU

汇编代码

```
;测试工作频率为11.0592MHz

P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     000BH
        LJMP    TM0ISR

        ORG     0100H
TM0ISR:
        CPL     P1.0            ;测试端口
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     TMOD,#00H       ;模式0
        MOV     TL0,#66H        ;65536-11.0592M/12/1000
        MOV     TH0,#0FCH
        SETB    TR0             ;启动定时器
        SETB    ET0             ;使能定时器中断
        SETB    EA

        JMP     $

        END
```

12.5.2 定时器 0（模式 1－16 位不自动重载），用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM0_Isr() interrupt 1
{
    TL0 = 0x66;                 //重设定时参数
    TH0 = 0xfc;
    P10 = !P10;                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x01;                //模式 1
    TL0 = 0x66;                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                    //启动定时器
    ET0 = 1;                    //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1        DATA        093H
P0M0        DATA        094H
```

```
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR

            ORG     0100H
TM0ISR:
            MOV     TL0,#66H            ;重设定时参数
            MOV     TH0,#0FCH
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#01H           ;模式 1
            MOV     TL0,#66H            ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0                 ;启动定时器
            SETB    ET0                 ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

12.5.3 定时器 0（模式 2—8 位自动重载），用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
```

```
#include "intrins.h"

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        P10         =   P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x02;                                //模式 2
    TL0 = 0xf4;                                 //256-11.0592M/12/76K
    TH0 = 0xf4;
    TR0 = 1;                                    //启动定时器
    ET0 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1        DATA        093H
P0M0        DATA        094H
P1M1        DATA        091H
P1M0        DATA        092H
P2M1        DATA        095H
P2M0        DATA        096H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
```

```
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR

            ORG     0100H
TM0ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#02H           ;模式2
            MOV     TL0,#0F4H           ;256-11.0592M/12/76K
            MOV     TH0,#0F4H
            SETB    TR0                 ;启动定时器
            SETB    ET0                 ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

12.5.4 定时器 0（模式 3－16 位自动重载不可屏蔽中断），用作定时

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
```

```
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x03;                                //模式 3
    TL0 = 0x66;                                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                    //启动定时器
    ET0 = 1;                                    //使能定时器中断
//  EA = 1;                                     //不受 EA 控制

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     000BH
```

```
            LJMP    TM0ISR

            ORG     0100H
TM0ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#03H           ;模式3
            MOV     TL0,#66H            ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0                 ;启动定时器
            SETB    ET0                 ;使能定时器中断
;           SETB    EA                  ;不受EA控制

            JMP     $

            END
```

12.5.5 定时器 0（外部计数－扩展T0 为外部下降沿中断）

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;
```

```
void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x04;                                //外部计数模式
    TL0 = 0xff;
    TH0 = 0xff;
    TR0 = 1;                                    //启动定时器
    ET0 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为11.0592MHz

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR

            ORG     0100H
TM0ISR:
            CPL     P1.0                    ;测试端口
            RETI

MAIN:
```

```
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     TMOD,#04H           ;外部计数模式
        MOV     TL0,#0FFH
        MOV     TH0,#0FFH
        SETB    TR0                 ;启动定时器
        SETB    ET0                 ;使能定时器中断
        SETB    EA

        JMP     $

        END
```

12.5.6 定时器 0（测量脉宽－INT0 高电平宽度）

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     AUXR    =   0x8e;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void INT0_Isr() interrupt 0
{
    P0 = TL0;                       //TL0 为测量值低字节
    P1 = TH0;                       //TH0 为测量值高字节
}

void main()
```

```
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    AUXR = 0x80;                    //1T 模式
    TMOD = 0x08;                    //使能 GATE,INT0 为 1 时使能计时
    TL0 = 0x00;
    TH0 = 0x00;
    while (INT0);                   //等待 INT0 为低
    TR0 = 1;                        //启动定时器
    IT0 = 1;                        //使能 INT0 下降沿中断
    EX0 = 1;
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR    DATA    8EH
P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     0003H
        LJMP    INT0ISR

        ORG     0100H
INT0ISR:
        MOV     P0,TL0          ;TL0 为测量值低字节
        MOV     P1,TH0          ;TH0 为测量值高字节
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
```

```
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     AUXR,#80H           ;1T 模式
        MOV     TMOD,#08H           ;使能 GATE,INT0 为 1 时使能计时
        MOV     TL0,#00H
        MOV     TH0,#00H
        JB      INT0,$              ;等待 INT0 为低
        SETB    TR0                 ;启动定时器
        SETB    IT0                 ;使能 INT0 下降沿中断
        SETB    EX0
        SETB    EA

        JMP     $

        END
```

12.5.7 定时器 0（模式 0），时钟分频输出

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     INTCLKO     =   0x8f;

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
```

```
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;                    //模式 0
    TL0 = 0x66;                     //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                        //启动定时器
    INTCLKO = 0x01;                 //使能时钟输出

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

INTCLKO     DATA    8FH
P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#00H       ;模式 0
            MOV     TL0,#66H        ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0             ;启动定时器
```

```
        MOV     INTCLKO,#01H            ;使能时钟输出
        JMP     $
        END
```

12.5.8 定时器 1（模式 0－16 位自动重载），用作定时

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM1_Isr() interrupt 3
{
    P10 = !P10;                             //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;                            //模式 0
    TL1 = 0x66;                             //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                                //启动定时器
    ET1 = 1;                                //使能定时器中断
    EA = 1;
```

STC MCU

```
    while (1);
}
```

汇编代码

```
;测试工作频率为11.0592MHz

P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     001BH
        LJMP    TM1ISR

        ORG     0100H
TM1ISR:
        CPL     P1.0            ;测试端口
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     TMOD,#00H       ;模式0
        MOV     TL1,#66H        ;65536-11.0592M/12/1000
        MOV     TH1,#0FCH
        SETB    TR1             ;启动定时器
        SETB    ET1             ;使能定时器中断
        SETB    EA

        JMP     $

        END
```

STC MCU

12.5.9 定时器 1（模式 1－16 位不自动重载），用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        P10         =   P1^0;

void TM1_Isr() interrupt 3
{
    TL1 = 0x66;                                 //重设定时参数
    TH1 = 0xfc;
    P10 = !P10;                                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x10;                                //模式 1
    TL1 = 0x66;                                 //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                                    //启动定时器
    ET1 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为11.0592MHz

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     001BH
            LJMP    TM1ISR

            ORG     0100H
TM1ISR:
            MOV     TL1,#66H            ;重设定时参数
            MOV     TH1,#0FCH
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#10H           ;模式1
            MOV     TL1,#66H            ;65536-11.0592M/12/1000
            MOV     TH1,#0FCH
            SETB    TR1                 ;启动定时器
            SETB    ET1                 ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

12.5.10 定时器 1（模式 2—8 位自动重载），用作定时

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        P10         =   P1^0;

void TM1_Isr() interrupt 3
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x20;                                //模式2
    TL1 = 0xf4;                                 //256-11.0592M/12/76K
    TH1 = 0xf4;
    TR1 = 1;                                    //启动定时器
    ET1 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为11.0592MHz

P0M1        DATA        093H
P0M0        DATA        094H
P1M1        DATA        091H
P1M0        DATA        092H
```

```
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     001BH
            LJMP    TM1ISR

            ORG     0100H
TM1ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#20H           ;模式2
            MOV     TL1,#0F4H           ;256-11.0592M/12/76K
            MOV     TH1,#0F4H
            SETB    TR1                 ;启动定时器
            SETB    ET1                 ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

12.5.11 定时器 1（外部计数－扩展T1 为外部下降沿中断）

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
```

```
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM1_Isr() interrupt 3
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x40;                                //外部计数模式
    TL1 = 0xff;
    TH1 = 0xff;
    TR1 = 1;                                    //启动定时器
    ET1 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH
```

```
                ORG         0000H
                LJMP        MAIN
                ORG         001BH
                LJMP        TM1ISR

                ORG         0100H
TM1ISR:
                CPL         P1.0                ;测试端口
                RETI

MAIN:
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                MOV         TMOD,#40H           ;外部计数模式
                MOV         TL1,#0FFH
                MOV         TH1,#0FFH
                SETB        TR1                 ;启动定时器
                SETB        ET1                 ;使能定时器中断
                SETB        EA

                JMP         $

                 END
```

12.5.12 定时器 1（测量脉宽－INT1 高电平宽度）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
```

```
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sfr     AUXR    =   0x8e;

void INT1_Isr() interrupt 2
{
    P0 = TL1;                       //TL1 为测量值低字节
    P1 = TH1;                       //TH1 为测量值高字节
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    AUXR = 0x40;                    //1T 模式
    TMOD = 0x80;                    //使能 GATE,INT1 为 1 时使能计时
    TL1 = 0x00;
    TH1 = 0x00;
    while (INT1);                   //等待 INT1 为低
    TR1 = 1;                        //启动定时器
    IT1 = 1;                        //使能 INT1 下降沿中断
    EX1 = 1;
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR    DATA    8EH
P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
```

```
                ORG         0013H
                LJMP        INT1ISR

                ORG         0100H
INT1ISR:
                MOV         P0,TL1              ;TL1 为测量值低字节
                MOV         P1,TH1              ;TH1 为测量值高字节
                RETI

MAIN:
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                MOV         AUXR,#40H           ;1T 模式
                MOV         TMOD,#80H           ;使能 GATE,INT1 为 1 时使能计时
                MOV         TL1,#00H
                MOV         TH1,#00H
                JB          INT1,$              ;等待 INT1 为低
                SETB        TR1                 ;启动定时器
                SETB        IT1                 ;使能 INT1 下降沿中断
                SETB        EX1
                SETB        EA

                JMP         $

                END
```

12.5.13 定时器 1（模式 0），时钟分频输出

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         INTCLKO     =   0x8f;

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
```

```
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;                    //模式 0
    TL1 = 0x66;                     //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                        //启动定时器
    INTCLKO = 0x02;                 //使能时钟输出

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

INTCLKO     DATA    8FH
P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
```

```
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     TMOD,#00H           ;模式 0
        MOV     TL1,#66H            ;65536-11.0592M/12/1000
        MOV     TH1,#0FCH
        SETB    TR1                 ;启动定时器
        MOV     INTCLKO,#02H        ;使能时钟輸出

        JMP     $

        END
```

12.5.14 定时器 1（模式 0）做串口 1 波特率发生器

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define FOSC        11059200UL
#define BRT         (65536 - FOSC / 115200 / 4)

sfr     AUXR    =   0x8e;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

bit     busy;
char    wptr;
char    rptr;
char    buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
```

```
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSENDStr("Uart Test !\r\n");

    while (1)
    {
```

STC MCU

```
        if (rptr != wptr)
        {
            UartSEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA    8EH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,SBUF
            INC     WPTR
UARTISR_EXIT:
            POP     PSW
            POP     ACC
```

STC MCU

```
                RETI

UART_INIT:
                MOV         SCON,#50H
                MOV         TMOD,#00H
                MOV         TL1,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV         TH1,#0FFH
                SETB        TR1
                MOV         AUXR,#40H
                CLR         BUSY
                MOV         WPTR,#00H
                MOV         RPTR,#00H
                RET

UART_SEND:
                JB          BUSY,$
                SETB        BUSY
                MOV         SBUF,A
                RET

UART_SENDSTR:
                CLR         A
                MOVC        A,@A+DPTR
                JZ          SENDEND
                LCALL       UART_SEND
                INC         DPTR
                JMP         UART_SENDSTR
SENDEND:
                RET

MAIN:
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                LCALL       UART_INIT
                SETB        ES
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
```

STC MCU

```
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

12.5.15 定时器 1（模式 2）做串口 1 波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define    FOSC         11059200UL
#define    BRT          (256 - FOSC / 115200 / 32)

sfr        AUXR         =    0x8e;

sfr        P0M1         =    0x93;
sfr        P0M0         =    0x94;
sfr        P1M1         =    0x91;
sfr        P1M0         =    0x92;
sfr        P2M1         =    0x95;
sfr        P2M0         =    0x96;
sfr        P3M1         =    0xb1;
sfr        P3M0         =    0xb2;
sfr        P4M1         =    0xb3;
sfr        P4M0         =    0xb4;
sfr        P5M1         =    0xc9;
sfr        P5M0         =    0xca;

bit        busy;
char       wptr;
char       rptr;
char       buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
```

STC MCU

```
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x20;
    TL1 = BRT;
    TH1 = BRT;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

STC MCU

汇编代码

```
;测试工作频率为11.0592MHz

AUXR        DATA    8EH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,SBUF
            INC     WPTR
UARTISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART_INIT:
            MOV     SCON,#50H
            MOV     TMOD,#20H
            MOV     TL1,#0FDH           ;256-11059200/115200/32=0FDH
            MOV     TH1,#0FDH
```

STC MCU

```
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

UART_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     SBUF,A
                RET

UART_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SENDEND
                LCALL   UART_SEND
                INC     DPTR
                JMP     UART_SENDSTR
SENDEND:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART_SEND
                INC     RPTR
                JMP     LOOP
```

STC MCU

```
STRING:     DB          'Uart Test !',0DH,0AH,00H

            END
```

12.5.16 定时器 2（16 位自动重载），用作定时

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         T2L         =   0xd7;
sfr         T2H         =   0xd6;
sfr         AUXR        =   0x8e;
sfr         IE2         =   0xaf;
#define     ET2             0x04
sfr         AUXINTIF    =   0xef;
#define     T2IF            0x01

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        P10         =   P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                     //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
```

STC MCU

```
    T2L = 0x66;                         //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                        //启动定时器
    IE2 = ET2;                          //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0063H
            LJMP    TM2ISR

            ORG     0100H
TM2ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H
```

STC MCU

```
        MOV     T2L,#66H                ;65536-11.0592M/12/1000
        MOV     T2H,#0FCH
        MOV     AUXR,#10H               ;启动定时器
        MOV     IE2,#ET2                ;使能定时器中断
        SETB    EA

        JMP     $

        END
```

12.5.17 定时器 2（外部计数一扩展T2 为外部下降沿中断）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2         0x04
sfr     AUXINTIF=   0xef;
#define T2IF        0x01

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;                         //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
```

```
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T2L = 0xff;
    T2H = 0xff;
    AUXR = 0x18;                    //设置外部计数模式并启动定时器
    IE2 = ET2;                      //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0063H
            LJMP    TM2ISR

            ORG     0100H
TM2ISR:
            CPL     P1.0            ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
```

STC MCU

```
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     T2L,#0FFH
        MOV     T2H,#0FFH
        MOV     AUXR,#18H       ;设置外部计数模式并启动定时器
        MOV     IE2,#ET2        ;使能定时器中断
        SETB    EA

        JMP     $

        END
```

12.5.18 定时器 2，时钟分频输出

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     INTCLKO =   0x8f;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
```

STC MCU

```
    P5M1 = 0x00;

    T2L = 0x66;                         //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                        //启动定时器
    INTCLKO = 0x04;                     //使能时钟输出

    while (1);
}
```

汇编代码

```
;测试工作频率为11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
INTCLKO     DATA    8FH

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     T2L,#66H            ;65536-11.0592M/12/1000
            MOV     T2H,#0FCH
            MOV     AUXR,#10H           ;启动定时器
            MOV     INTCLKO,#04H        ;使能时钟输出

            JMP     $

            END
```

STC MCU

12.5.19 定时器 2 做串口 1 波特率发生器

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;
sfr       T2H         =   0xd6;
sfr       T2L         =   0xd7;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x15;
```

STC MCU

```
        wptr = 0x00;
        rptr = 0x00;
        busy = 0;
}

void UartSend(char dat)
{
        while (busy);
        busy = 1;
        SBUF = dat;
}

void UartSendStr(char *p)
{
        while (*p)
        {
                UartSEND(*p++);
        }
}

void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        UartInit();
        ES = 1;
        EA = 1;
        UartSENDStr("Uart Test !\r\n");

        while (1)
        {
                if (rptr != wptr)
                {
                        UartSEND(buffer[rptr++]);
                        rptr &= 0x0f;
                }
        }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA        8EH
T2H         DATA        0D6H
T2L         DATA        0D7H

BUSY        BIT         20H.0
```

```
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,SBUF
            INC     WPTR
UARTISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART_INIT:
            MOV     SCON,#50H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#15H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART_SEND:
            JB      BUSY,$
            SETB    BUSY
```

STC MCU

```
                MOV     SBUF,A
                RET

UART_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SENDEND
                LCALL   UART_SEND
                INC     DPTR
                JMP     UART_SENDSTR
SENDEND:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

12.5.20 定时器 2 做串口 2 波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;
sfr       T2H         =   0xd6;
sfr       T2L         =   0xd7;
sfr       S2CON       =   0x9a;
sfr       S2BUF       =   0x9b;
sfr       IE2         =   0xaf;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart2Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;
        busy = 0;
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f;
    }
}

void Uart2Init()
{
    S2CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

STC MCU

```
void Uart2Send(char dat)
{
    while (busy);
    busy = 1;
    S2BUF = dat;
}

void Uart2SendStr(char *p)
{
    while (*p)
    {
        Uart2SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart2SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR    DATA    8EH
T2H     DATA    0D6H
T2L     DATA    0D7H
S2CON   DATA    9AH
S2BUF   DATA    9BH
IE2     DATA    0AFH

BUSY    BIT     20H.0
WPTR    DATA    21H
```

```
RPTR        DATA    22H
BUFFER      DATA    23H             ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0043H
            LJMP    UART2_ISR

            ORG     0100H

UART2_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S2CON
            JNB     ACC.1,CHKRI
            ANL     S2CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART2ISR_EXIT
            ANL     S2CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S2BUF
            INC     WPTR
UART2ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART2_INIT:
            MOV     S2CON,#10H
            MOV     T2L,#0E8H       ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART2_SEND:
            JB      BUSY,$
            SETB    BUSY
```

```
                MOV         S2BUF,A
                RET

UART2_SENDSTR:
                CLR         A
                MOVC        A,@A+DPTR
                JZ          SEND2END
                LCALL       UART2_SEND
                INC         DPTR
                JMP         UART2_SENDSTR
SEND2END:
                RET

MAIN:
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                LCALL       UART2_INIT
                MOV         IE2,#01H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART2_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART2_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

12.5.21 定时器 2 做串口 3 波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;
sfr       T2H         =   0xd6;
sfr       T2L         =   0xd7;
sfr       S3CON       =   0xac;
sfr       S3BUF       =   0xad;
sfr       IE2         =   0xaf;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

void Uart3Init()
{
    S3CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

STC MCU

```
void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA        8EH
T2H         DATA        0D6H
T2L         DATA        0D7H
S3CON       DATA        0ACH
S3BUF       DATA        0ADH
IE2         DATA        0AFH

BUSY        BIT         20H.0
WPTR        DATA        21H
```

```
RPTR        DATA    22H
BUFFER      DATA    23H             ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3_ISR

            ORG     0100H

UART3_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S3CON
            JNB     ACC.1,CHKRI
            ANL     S3CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART3ISR_EXIT
            ANL     S3CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S3BUF
            INC     WPTR
UART3ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART3_INIT:
            MOV     S3CON,#10H
            MOV     T2L,#0E8H       ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART3_SEND:
            JB      BUSY,$
            SETB    BUSY
```

```
                MOV         S3BUF,A
                RET

UART3_SENDSTR:
                CLR         A
                MOVC        A,@A+DPTR
                JZ          SEND3END
                LCALL       UART3_SEND
                INC         DPTR
                JMP         UART3_SENDSTR
SEND3END:
                RET

MAIN:
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                LCALL       UART3_INIT
                MOV         IE2,#08H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART3_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART3_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

STC MCU

12.5.22 定时器 2 做串口 4 波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;
sfr       T2H         =   0xd6;
sfr       T2L         =   0xd7;
sfr       S4CON       =   0x84;
sfr       S4BUF       =   0x85;
sfr       IE2         =   0xaf;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

STC MCU

```
void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR    DATA    8EH
T2H     DATA    0D6H
T2L     DATA    0D7H
S4CON   DATA    84H
S4BUF   DATA    85H
IE2     DATA    0AFH

BUSY    BIT     20H.0
WPTR    DATA    21H
```

```
RPTR        DATA    22H
BUFFER      DATA    23H             ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0093H
            LJMP    UART4_ISR

            ORG     0100H

UART4_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S4CON
            JNB     ACC.1,CHKRI
            ANL     S4CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART4ISR_EXIT
            ANL     S4CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S4BUF
            INC     WPTR
UART4ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART4_INIT:
            MOV     S4CON,#10H
            MOV     T2L,#0E8H       ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART4_SEND:
            JB      BUSY,$
            SETB    BUSY
```

```
                MOV         S4BUF,A
                RET

UART4_SENDSTR:
                CLR         A
                MOVC        A,@A+DPTR
                JZ          SEND4END
                LCALL       UART4_SEND
                INC         DPTR
                JMP         UART4_SENDSTR
SEND4END:
                RET

MAIN:
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                LCALL       UART4_INIT
                MOV         IE2,#10H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART4_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART4_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

STC MCU

12.5.23 定时器 3（16 位自动重载），用作定时

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         T4T3M       =   0xd1;
sfr         T4L         =   0xd3;
sfr         T4H         =   0xd2;
sfr         T3L         =   0xd5;
sfr         T3H         =   0xd4;
sfr         T2L         =   0xd7;
sfr         T2H         =   0xd6;
sfr         AUXR        =   0x8e;
sfr         IE2         =   0xaf;
#define     ET2             0x04
#define     ET3             0x20
#define     ET4             0x40
sfr         AUXINTIF    =   0xef;
#define     T2IF            0x01
#define     T3IF            0x02
#define     T4IF            0x04

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        P10         =   P1^0;

void TM3_Isr() interrupt 19
{
    P10 = !P10;                     //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T3L = 0x66;                     //65536-11.0592M/12/1000
```

STC MCU

```
    T3H = 0xfc;
    T4T3M = 0x08;                   //启动定时器
    IE2 = ET3;                      //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
ET3         EQU     20H
ET4         EQU     40H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H
T3IF        EQU     02H
T4IF        EQU     04H

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     009BH
            LJMP    TM3ISR

            ORG     0100H
TM3ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
```

STC MCU

```
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     T3L,#66H                ;65536-11.0592M/12/1000
        MOV     T3H,#0FCH
        MOV     T4T3M,#08H              ;启动定时器
        MOV     IE2,#ET3                ;使能定时器中断
        SETB    EA

        JMP     $

        END
```

12.5.24 定时器 3（外部计数－扩展T3 为外部下降沿中断）

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2         0x04
#define ET3         0x20
#define ET4         0x40
sfr     AUXINTIF=   0xef;
#define T2IF        0x01
#define T3IF        0x02
#define T4IF        0x04

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;
```

```
sbit        P10             =     P1^0;

void TM3_Isr() interrupt 19
{
    P10 = !P10;                         //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T3L = 0x66;                         //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x0c;                       //设置外部计数模式并启动定时器
    IE2 = ET3;                          //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为11.0592MHz
T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
ET3         EQU     20H
ET4         EQU     40H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H
T3IF        EQU     02H
T4IF        EQU     04H

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
```

```
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     009BH
            LJMP    TM3ISR

            ORG     0100H
TM3ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     T3L,#66H            ;65536-11.0592M/12/1000
            MOV     T3H,#0FCH
            MOV     T4T3M,#0CH          ;设置外部计数模式并启动定时器
            MOV     IE2,#ET3            ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

12.5.25 定时器 3，时钟分频输出

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
```

```
sfr     T2H     =   0xd6;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T3L = 0x66;                 //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x09;               //使能时钟输出并启动定时器

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T4T3M   DATA    0D1H
T4L     DATA    0D3H
T4H     DATA    0D2H
T3L     DATA    0D5H
T3H     DATA    0D4H
T2L     DATA    0D7H
T2H     DATA    0D6H

P1M1    DATA    091H
P1M0    DATA    092H
P0M1    DATA    093H
P0M0    DATA    094H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
```

```
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     T3L,#66H            ;65536-11.0592M/12/1000
            MOV     T3H,#0FCH
            MOV     T4T3M,#09H          ;使能时钟输出并启动定时器

            JMP     $

            END
```

12.5.26 定时器 3 做串口 3 波特率发生器

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC      11059200UL
#define   BRT       (65536 - FOSC / 115200 / 4)

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     S3CON   =   0xac;
sfr     S3BUF   =   0xad;
sfr     IE2     =   0xaf;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
```

```
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

bit     busy;
char    wptr;
char    rptr;
char    buffer[16];

void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

void Uart3Init()
{
    S3CON = 0x50;
    T3L = BRT;
    T3H = BRT >> 8;
    T4T3M = 0x0a;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
```

STC MCU

```
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH
```

```
                ORG     0000H
                LJMP    MAIN
                ORG     008BH
                LJMP    UART3_ISR

                ORG     0100H

UART3_ISR:
                PUSH    ACC
                PUSH    PSW
                MOV     PSW,#08H

                MOV     A,S3CON
                JNB     ACC.1,CHKRI
                ANL     S3CON,#NOT 02H
                CLR     BUSY
CHKRI:
                JNB     ACC.0,UART3ISR_EXIT
                ANL     S3CON,#NOT 01H
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,S3BUF
                INC     WPTR
UART3ISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART3_INIT:
                MOV     S3CON,#50H
                MOV     T3L,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     T3H,#0FFH
                MOV     T4T3M,#0AH
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

UART3_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     S3BUF,A
                RET

UART3_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SEND3END
                LCALL   UART3_SEND
                INC     DPTR
                JMP     UART3_SENDSTR
SEND3END:
                RET

MAIN:
                MOV     SP, #5FH
```

```
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                LCALL       UART3_INIT
                MOV         IE2,#08H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART3_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART3_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

12.5.27 定时器 4（16 位自动重载），用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         T4T3M       =   0xd1;
sfr         T4L         =   0xd3;
sfr         T4H         =   0xd2;
sfr         T3L         =   0xd5;
sfr         T3H         =   0xd4;
sfr         T2L         =   0xd7;
sfr         T2H         =   0xd6;
sfr         AUXR        =   0x8e;
sfr         IE2         =   0xaf;
```

```
#define   ET2             0x04
#define   ET3             0x20
#define   ET4             0x40
sfr       AUXINTIF    =   0xef;
#define   T2IF            0x01
#define   T3IF            0x02
#define   T4IF            0x04

sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

sbit      P10         =   P1^0;

void TM4_Isr() interrupt 20
{
    P10 = !P10;                     //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T4L = 0x66;                     //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x80;                   //启动定时器
    IE2 = ET4;                      //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为11.0592MHz

T4T3M       DATA        0D1H
T4L         DATA        0D3H
T4H         DATA        0D2H
```

```
T3L     DATA    0D5H
T3H     DATA    0D4H
T2L     DATA    0D7H
T2H     DATA    0D6H
AUXR    DATA    8EH
IE2     DATA    0AFH
ET2     EQU     04H
ET3     EQU     20H
ET4     EQU     40H
AUXINTIF DATA   0EFH
T2IF    EQU     01H
T3IF    EQU     02H
T4IF    EQU     04H

P1M1    DATA    091H
P1M0    DATA    092H
P0M1    DATA    093H
P0M0    DATA    094H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     00A3H
        LJMP    TM4ISR

        ORG     0100H
TM4ISR:
        CPL     P1.0            ;测试端口
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     T4L,#66H        ;65536-11.0592M/12/1000
        MOV     T4H,#0FCH
        MOV     T4T3M,#80H      ;启动定时器
        MOV     IE2,#ET4        ;使能定时器中断
        SETB    EA

        JMP     $
```

STC MCU

```
END
```

12.5.28 定时器 4（外部计数一扩展T4 为外部下降沿中断）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2     0x04
#define ET3     0x20
#define ET4     0x40
sfr     AUXINTIF=   0xef;
#define T2IF    0x01
#define T3IF    0x02
#define T4IF    0x04

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM4_Isr() interrupt 20
{
    P10 = !P10;                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
```

```
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T4L = 0x66;                 //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0xc0;               //设置外部计数模式并启动定时器
    IE2 = ET4;                  //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
ET3         EQU     20H
ET4         EQU     40H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H
T3IF        EQU     02H
T4IF        EQU     04H

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00A3H
            LJMP    TM4ISR

            ORG     0100H
TM4ISR:
            CPL     P1.0                ;测试端口
```

STC MCU

```
            RETI

MAIN:
            MOV         SP, #5FH
            MOV         P0M0, #00H
            MOV         P0M1, #00H
            MOV         P1M0, #00H
            MOV         P1M1, #00H
            MOV         P2M0, #00H
            MOV         P2M1, #00H
            MOV         P3M0, #00H
            MOV         P3M1, #00H
            MOV         P4M0, #00H
            MOV         P4M1, #00H
            MOV         P5M0, #00H
            MOV         P5M1, #00H

            MOV         T4L,#66H                ;65536-11.0592M/12/1000
            MOV         T4H,#0FCH
            MOV         T4T3M,#0C0H             ;设置外部计数模式并启动定时器
            MOV         IE2,#ET4                ;使能定时器中断
            SETB        EA

            JMP         $

            END
```

12.5.29 定时器 4，时钟分频输出

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;
```

```
void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P3M1 = 0x00;

    T4L = 0x66;                         //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x90;                       //使能时钟输出并启动定时器

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
```

```
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     T4L,#66H                ;65536-11.0592M/12/1000
        MOV     T4H,#0FCH
        MOV     T4T3M,#90H              ;使能时钟輸出并启动定时器

        JMP     $

        END
```

12.5.30 定时器 4 做串口 4 波特率发生器

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define FOSC 11059200UL
#define BRT (65536 - FOSC / 115200 / 4)

sfr T4T3M = 0xd1;
sfr T4L = 0xd3;
sfr T4H = 0xd2;
sfr T3L = 0xd5;
sfr T3H = 0xd4;
sfr T2L = 0xd7;
sfr T2H = 0xd6;
sfr S4CON = 0x84;
sfr S4BUF = 0x85;
sfr IE2 = 0xaf;

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

bit busy;
char wptr;
char rptr;
char buffer[16];
```

STC MCU

```
void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x50;
    T4L = BRT;
    T4H = BRT >> 8;
    T4T3M = 0xa0;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart4Init();
    IE2 = 0x10;
```

STC MCU

```
    EA = 1;
    Uart4SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
S4CON       DATA    84H
S4BUF       DATA    85H
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H             ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0093H
            LJMP    UART4_ISR

            ORG     0100H

UART4_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S4CON
```

STC MCU

```
                JNB     ACC.1,CHKRI
                ANL     S4CON,#NOT 02H
                CLR     BUSY
CHKRI:
                JNB     ACC.0,UART4ISR_EXIT
                ANL     S4CON,#NOT 01H
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,S4BUF
                INC     WPTR
UART4ISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART4_INIT:
                MOV     S4CON,#50H
                MOV     T4L,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     T4H,#0FFH
                MOV     T4T3M,#0A0H
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

UART4_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     S4BUF,A
                RET

UART4_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SEND4END
                LCALL   UART4_SEND
                INC     DPTR
                JMP     UART4_SENDSTR
SEND4END:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART4_INIT
```

STC MCU

```
        MOV     IE2,#10H
        SETB    EA

        MOV     DPTR,#STRING
        LCALL   UART4_SENDSTR

LOOP:
        MOV     A,RPTR
        XRL     A,WPTR
        ANL     A,#0FH
        JZ      LOOP
        MOV     A,RPTR
        ANL     A,#0FH
        ADD     A,#BUFFER
        MOV     R0,A
        MOV     A,@R0
        LCALL   UART4_SEND
        INC     RPTR
        JMP     LOOP

STRING: DB      'Uart Test !',0DH,0AH,00H

        END
```

STC MCU

13 串口通信

特别注意：每组串口的接收脚可设置为准双向口模式或者输入模式，串口的发送脚可设置为准双向口模式或者推挽输出模式，但发送脚一定不能设置开漏模式，否则没有串口信号输出

STC8A/8F 系列单片机具有 4 个全双工异步串行通信接口。每个串行口由 2 个数据缓冲器、一个移位寄存器、一个串行控制寄存器和一个波特率发生器等组成。每个串行口的数据缓冲器由 2 个互相独立的接收、发送缓冲器构成，可以同时发送和接收数据。

STC8A/8F 系列单片机的串口 1 有 4 种工作方式，其中两种方式的波特率是可变的，另两种是固定的，以供不同应用场合选用。串口 2/串口 3/串口 4 都只有两种工作方式，这两种方式的波特率都是可变的。用户可用软件设置不同的波特率和选择不同的工作方式。主机可通过查询或中断方式对接收/发送进行程序处理，使用十分灵活。

串口 1、串口 2、串口 3、串口 4 的通讯口均可以通过功能管脚的切换功能切换到多组端口，从而可以将一个通讯口分时复用为多个通讯口。

13.1 串口相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	串口 1 数据寄存器	99H									0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S2BUF	串口 2 数据寄存器	9BH									0000,0000
S3CON	串口 3 控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	串口 3 数据寄存器	ADH									0000,0000
S4CON	串口 4 控制寄存器	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	串口 4 数据寄存器	85H									0000,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
SADDR	串口 1 从机地址寄存器	A9H									0000,0000
SADEN	串口 1 从机地址屏蔽寄存器	B9H									0000,0000

13.2 串口 1

13.2.1 串口 1 控制寄存器（SCON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

SM0/FE：当PCON寄存器中的SMOD0位为1时，该位为帧错误检测标志位。当UART在接收过程中检测到一个无效停止位时，通过UART接收器将该位置1，必须由软件清零。当PCON寄存器中的SMOD0位为0时，该位和SM1一起指定串口1的通信工作模式，如下表所示：

SM0	SM1	串口1工作模式	功能说明
0	0	模式0	同步移位串行方式
0	1	模式1	可变波特率8位数据方式
1	0	模式2	固定波特率9位数据方式
1	1	模式3	可变波特率9位数据方式

SM2：允许模式 2 或模式 3 多机通信控制位。当串口 1 使用模式 2 或模式 3 时，如果 SM2 位为 1 且 REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 RB8）来筛选地址帧，若 RB8=1，说明该帧是地址帧，地址信息可以进入 SBUF，并使 RI 为 1，进而在中断服务程序中再进行地址号比较；若 RB8=0，说明该帧不是地址帧，应丢掉且保持 RI=0。在模式 2 或模式 3 中，如果 SM2 位为 0 且 REN 位为 1，接收收机处于地址帧筛选被禁止状态，不论收到的 RB8 为 0 或 1，均可使接收到的信息进入 SBUF，并使 RI=1，此时 RB8 通常为校验位。模式 1 和模式 0 为非多机通信方式，在这两种方式时，SM2 应设置为 0。

REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

TB8：当串口 1 使用模式 2 或模式 3 时，TB8 为要发送的第 9 位数据，按需要由软件置位或清 0。在模式 0 和模式 1 中，该位不用。

RB8：当串口 1 使用模式 2 或模式 3 时，RB8 为接收到的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位。在模式 0 和模式 1 中，该位不用。

TI：串口 1 发送中断请求标志位。在模式 0 中，当串口发送数据第 8 位结束时，由硬件自动将 TI 置 1，向主机请求中断，响应中断后 TI 必须用软件清零。在其他模式中，则在停止位开始发送时由硬件自动将 TI 置 1，向 CPU 发请求中断，响应中断后 TI 必须用软件清零。

RI：串口 1 接收中断请求标志位。在模式 0 中，当串口接收第 8 位数据结束时，由硬件自动将 RI 置 1，向主机请求中断，响应中断后 RI 必须用软件清零。在其他模式中，串行接收到停止位的中间时刻由硬件自动将 RI 置 1，向 CPU 发中断申请，响应中断后 RI 必须由软件清零。

13.2.2 串口 1 数据寄存器（SBUF）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SBUF	99H								

SBUF：串口 1 数据接收/发送缓冲区。SBUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 SBUF 进行读操作，实际是读取串口接收缓冲区，对 SBUF 进行写操作则是触发串口开始发送数据。

13.2.3 电源管理寄存器（PCON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

SMOD：串口 1 波特率控制位

0：串口 1 的各个模式的波特率都不加倍

1：串口 1 模式 1、模式 2、模式 3 的波特率加倍

SMOD0：帧错误检测控制位

0：无帧错检测功能

1：使能帧错误检测功能。此时 SCON 的 SM0/FE 为 FE 功能，即为帧错误检测标志位。

13.2.4 辅助寄存器 1（AUXR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

UART_M0x6：串口 1 模式 0 的通讯速度控制

0：串口 1 模式 0 的波特率不加倍，固定为 Fosc/12

1：串口 1 模式 0 的波特率 6 倍速，即固定为 Fosc/12*6 ＝ Fosc/2

S1ST2：串口 1 波特率发射器选择位

0：选择定时器 1 作为波特率发射器

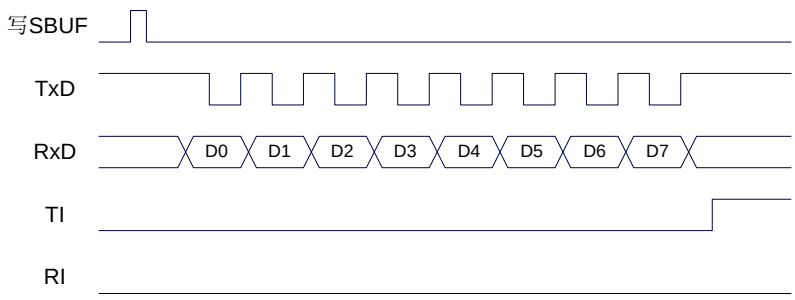
1：选择定时器 2 作为波特率发射器

13.2.5 串口 1 模式 0，模式 0 波特率计算公式

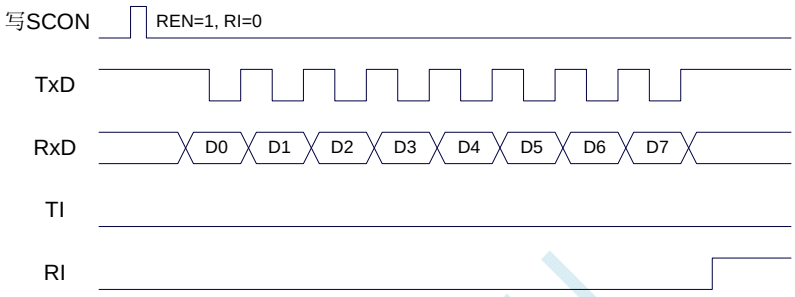
当串口 1 选择工作模式为模式 0 时，串行通信接口工作在同步移位寄存器模式，当串行口模式 0 的通信速度设置位 UART_M0x6 为 0 时，其波特率固定为系统时钟时钟的 12 分频（SYSclk/12）；当设置 UART_M0x6 为 1 时，其波特率固定为系统时钟频率的 2 分频（SYSclk/2）。RxD 为串行通讯的数据口，TxD 为同步移位脉冲输出脚，发送、接收的是 8 位数据，低位在先。

模式 0 的发送过程：当主机执行将数据写入发送缓冲器 SBUF 指令时启动发送，串行口即将 8 位数据以 SYSclk/12 或 SYSclk/2（由 UART_M0x6 确定是 12 分频还是 2 分频）的波特率从 RxD 管脚输出(从低位到高位)，发送完中断标志 TI 置 1，TxD 管脚输出同步移位脉冲信号。当写信号有效后，相隔一个时钟，发送控制端 SEND 有效(高电平)，允许 RxD 发送数据，同时允许 TxD 输出同步移位脉冲。一帧(8 位)数据发送完毕时，各控制端均恢复原状态，只有 TI 保持高电平，呈中断申请状态。在再次发送数据前，必须用软件将 TI 清 0。

模式 0 的接收过程：首先将接收中断请求标志 RI 清零并置位允许接收控制位 REN 时启动模式 0 接收过程。启动接收过程后，RxD 为串行数据输入端，TxD 为同步脉冲输出端。串行接收的波特率为 SYSclk/12 或 SYSclk/2（由 UART_M0x6 确定是 12 分频还是 2 分频）。当接收完成一帧数据（8 位）后，控制信号复位，中断标志 RI 被置 1，呈中断申请状态。当再次接收时，必须通过软件将 RI 清 0



发送数据（串口1模式0）



接收数据（串口1模式0）

工作于模式 0 时，必须清 0 多机通信控制位 SM2，使之不影响 TB8 位和 RB8 位。由于波特率固定为 SYSclk/12 或 SYSclk/2，无需定时器提供，直接由单片机的时钟作为同步移位脉冲。

串口 1 模式 0 的波特率计算公式如下表所示（SYSclk 为系统工作频率）：

UART_M0x6	波特率计算公式
0	波特率 $= \frac{SYSclk}{12}$
1	波特率 $= \frac{SYSclk}{2}$

13.2.6 串口 1 模式 1，模式 1 波特率计算公式

当软件设置 SCON 的 SM0、SM1 为“01”时，串行口 1 则以模式 1 进行工作。此模式为 8 位 UART 格式，一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，即可根据需要进行设置波特率。TxD 为数据发送口，RxD 为数据接收口，串行口全双工接受/发送。

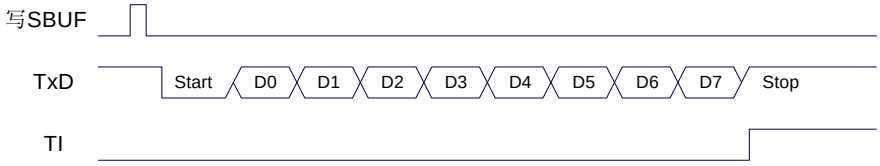
模式 1 的发送过程：串行通信模式发送时，数据由串行发送端 TxD 输出。当主机执行一条写 SBUF 的指令就启动串行通信的发送，写“SBUF”信号还把“1”装入发送移位寄存器的第 9 位，并通知 TX 控制单元开始发送。移位寄存器将数据不断右移送 TxD 端口发送，在数据的左边不断移入“0”作补充。当数据的最高位移到移位寄存器的输出位置，紧跟其后的是第 9 位“1”，在它的左边各位全为“0”，这个状态条件，使 TX 控制单元作最后一次移位输出，然后使允许发送信号“SEND”失效，完成一帧信息的发送，并置位中断请求位 TI，即 TI=1，向主机请求中断处理。

模式 1 的接收过程：当软件置位接收允许标志位 REN，即 REN=1 时，接收器便对 RxD 端口的信号进行检测，当检测到 RxD 端口发送从“1”→“0”的下降沿跳变时就启动接收器准备接收数据，并立即复位波特率发生器的接收计数器，将 1FFH 装入移位寄存器。接收的数据从接收移位寄存器的右边移入，已装入的 1FFH 向左边移出，当起始位"0"移到移位寄存器的最左边时，使 RX 控制器作最后一次移位，完成一帧的接收。若同时满足以下两个条件：

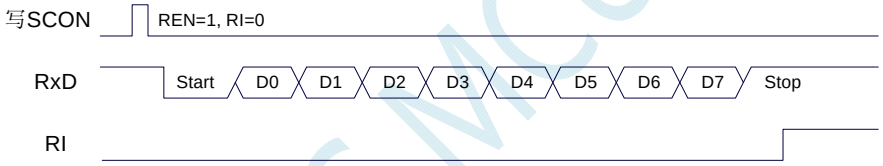
• RI=0；

• SM2=0 或接收到的停止位为 1。

则接收到的数据有效，实现装载入 SBUF，停止位进入 RB8，RI 标志位被置 1，向主机请求中断，若上述两条件不能同时满足，则接收到的数据作废并丢失，无论条件满足与否，接收器重又检测 RxD 端口上的"1"→"0"的跳变，继续下一帧的接收。接收有效，在响应中断后，RI 标志位必须由软件清 0。通常情况下，串行通信工作于模式 1 时，SM2 设置为"0"。



发送数据（串口1模式1）



接收数据（串口1模式1）

串口 1 的波特率是可变的，其波特率可由定时器 1 或者定时器 2 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 1 模式 1 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536-\frac{SYSclk}{4\times 波特率}$
	12T	定时器2重载值 = $65536-\frac{SYSclk}{12\times 4\times 波特率}$
定时器1模式0	1T	定时器1重载值 = $65536-\frac{SYSclk}{4\times 波特率}$
	12T	定时器1重载值 = $65536-\frac{SYSclk}{12\times 4\times 波特率}$
定时器1模式2	1T	定时器1重载值 = $256-\frac{2^{SMOD}\times SYSclk}{32\times 波特率}$
	12T	定时器1重载值 = $256-\frac{2^{SMOD}\times SYSclk}{12\times 32\times 波特率}$

下面为常用频率与常用波特率所对应定时器的重载值

频率 (MHz)	波特率	定时器 2		定时器 1 模式 0		定时器 1 模式 2			
						SMOD=1		SMOD=0	
		1T 模式	12T 模式	1T 模式	12T 模式	1T 模式	12T 模式	1T 模式	12T 模式
11.0592	115200	FFE8H	FFFEH	FFE8H	FFFEH	FAH	-	FDH	-
	57600	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	38400	FFB8H	FFFAH	FFB8H	FFFAH	EEH	-	F7H	-
	19200	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	9600	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
18.432	115200	FFD8H	-	FFD8H	-	F6H	-	FBH	-
	57600	FFB0H	-	FFB0H	-	ECH	-	F6H	-
	38400	FF88H	FFF6H	FF88H	FFF6H	E2H	-	F1H	-
	19200	FF10H	FFECH	FF10H	FFECH	C4H	FBH	E2H	-
	9600	FE20H	FFD8H	FE20H	FFD8H	88H	F6H	C4H	FBH
22.1184	115200	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	57600	FFA0H	FFF8H	FFA0H	FFF8H	E8H	FEH	F4H	FFH
	38400	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	19200	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
	9600	FDC0H	FFD0H	FDC0H	FFD0H	70H	F4H	B8H	FAH

13.2.7 串口 1 模式 2，模式 2 波特率计算公式

当 SM0、SM1 两位为 10 时，串行口 1 工作在模式 2。串行口 1 工作模式 2 为 9 位数据异步通信 UART 模式，其一帧的信息由 11 位组成：1 位起始位，8 位数据位（低位在先），1 位可编程位（第 9 位数据）和 1 位停止位。发送时可编程位（第 9 位数据）由 SCON 中的 TB8 提供，可软件设置为 1 或 0，或者可将 PSW 中的奇/偶校验位 P 值装入 TB8（TB8 既可作为多机通信中的地址数据标志位，又可作为数据的奇偶校验位）。接收时第 9 位数据装入 SCON 的 RB8。TxD 为发送端口，RxD 为接收端口，以全双工模式进行接收/发送。

模式 2 的波特率固定为系统时钟的 64 分频或 32 分频（取决于 PCON 中 SMOD 的值）

串口 1 模式 2 的波特率计算公式如下表所示（SYSclk 为系统工作频率）：

SMOD	波特率计算公式
0	波特率 $= \frac{SYSclk}{64}$
1	波特率 $= \frac{SYSclk}{32}$

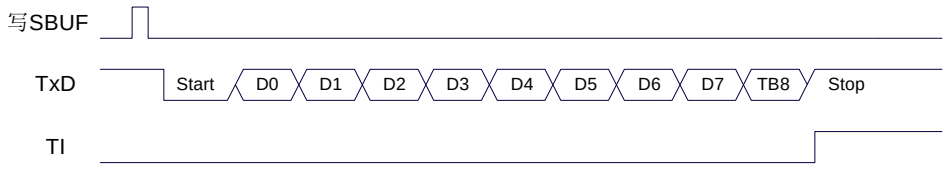
模式 2 和模式 1 相比，除波特率发生源略有不同，发送时由 TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收/发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件：

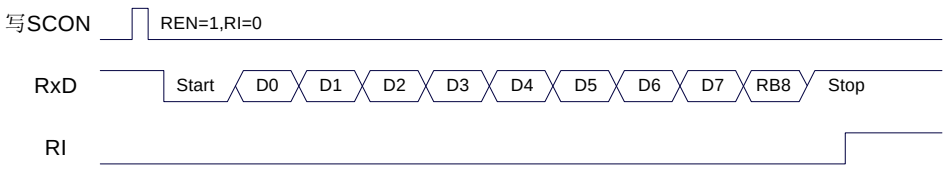
- RI=0
- SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中，RI 标志位被置 1，并向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 RI。无论上述条件满足与否，接收器又重新开始检测 RxD 输入端口的跳变信息，接收下一帧的输入信息。在模式 2 中，接收到的停止位与 SBUF、RB8 和 RI 无关。

通过软件对 SCON 中的 SM2、TB8 的设置以及通信 Ð 议的约定，为多机通信提供了方便。



发送数据（串口1模式2）



接收数据（串口1模式2）

13.2.8 串口 1 模式 3，模式 3 波特率计算公式

当 SM0、SM1 两位为 11 时，串行口 1 工作在模式 3。串行通信模式 3 为 9 位数据异步通信 UART

模式，其一帧的信息由 11 位组成：1 位起始位，8 位数据位（低位在先），1 位可编程位（第 9 位数据）和 1 位停止位。发送时可编程位（第 9 位数据）由 SCON 中的 TB8 提供，可软件设置为 1 或 0，或者可将 PSW 中的奇/偶校验位 P 值装入 TB8（TB8 既可作为多机通信中的地址数据标志位，又可作为数据的奇偶校验位）。接收时第 9 位数据装入 SCON 的 RB8。TxD 为发送端口，RxD 为接收端口，以全双工模式进行接收/发送。

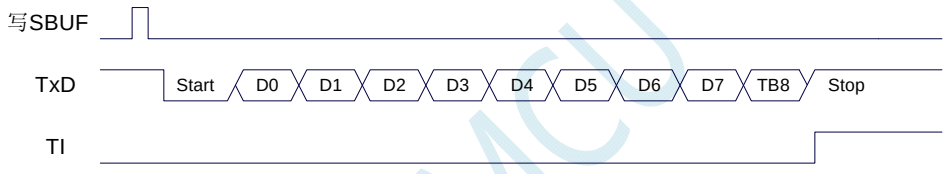
模式 3 和模式 1 相比，除发送时由 TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收‘发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件：

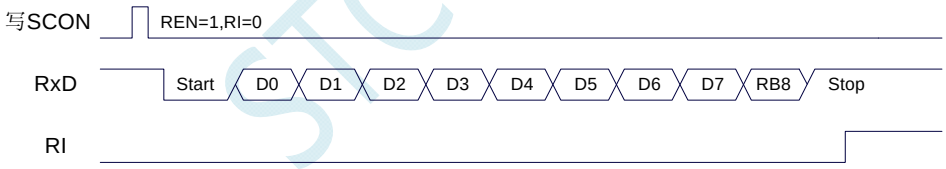
- RI=0
- SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中，RI 标志位被置 1，并向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 RI。无论上述条件满足与否，接收器又重新开始检测 RxD 输入端口的跳变信息，接收下一帧的输入信息。在模式 3 中，接收到的停止位与 SBUF、RB8 和 RI 无关。

通过软件对 SCON 中的 SM2、TB8 的设置以及通信协议的约定，为多机通信提供了方便。



发送数据（串口1模式3）



接收数据（串口1模式3）

串口 1 模式 3 的波特率计算公式与模式 1 是完全相同的。请参考模式 1 的波特率计算公式。

13.2.9 自动地址识别

13.2.10串口 1 从机地址控制寄存器（SADDR，SADEN）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SADDR	A9H								
SADEN	B9H								

SADDR：从机地址寄存器

SADEN：从机地址屏蔽位寄存器

自动地址识别功能典型应用在多机通讯领域，其主要原理是从机系统通过硬件比较功能来识别来自于主机串口数据流中的地址信息，通过寄存器 SADDR 和 SADEN 设置的本机的从机地址，硬件自动对从机地址进行过滤，当来自于主机的从机地址信息与本机所设置的从机地址相匹配时，硬件产生串口中

断；否则硬件自动丢弃串口数据，而不产生中断。当众多处于空闲模式的从机链接在一起时，只有从机地址相匹配的从机才会从空闲模式唤醒，从而可以大大降低从机 MCU 的功耗，即使从机处于正常工作状态也可避免不停地进入串口中断而降低系统执行效率。

要使用串口的自动地址识别功能，首先需要将参与通讯的 MCU 的串口通讯模式设置为模式 2 或者模式 3（通常都选择波特率可变的模式 3，因为模式 2 的波特率是固定的，不便于调节），并开启从机的 SCON 的 SM2 位。对于串口模式 2 或者模式 3 的 9 位数据位中，第 9 位数据（存放在 RB8 中）为地址/数据的标志位，当第 9 位数据为 1 时，表示前面的 8 位数据（存放在 SBUF 中）为地址信息。当 SM2 被设置为 1 时，从机 MCU 会自动过滤掉非地址数据（第 9 位为 0 的数据），而对 SBUF 中的地址数据（第 9 位为 1 的数据）自动与 SADDR 和 SADEN 所设置的本机地址进行比较，若地址相匹配，则会将 RI 置“1”，并产生中断，否则不予处理本次接收的串口数据。

从机地址的设置是通过 SADDR 和 SADEN 两个寄存器进行设置的。SADDR 为从机地址寄存器，里面存放本机的从机地址。SADEN 为从机地址屏蔽位寄存器，用于设置地址信息中的忽略位，设置方法如下：

例如
SADDR ＝ 11001010
SADEN ＝ 10000001
则匹配地址为 1xxxxxx0
即，只要主机送出的地址数据中的 bit0 为 0 且 bit7 为 1 就可以和本机地址相匹配

再例如
SADDR ＝ 11001010
SADEN ＝ 00001111
则匹配地址为 xxxx1010
即，只要主机送出的地址数据中的低 4 位为 1010 就可以和本机地址相匹配，而高 4 为被忽略，可以为任意值。

主机可以使用广播地址（FFH）同时选中所有的从机来进行通讯。

STC MCU

13.3 串口 2

13.3.1 串口 2 控制寄存器（S2CON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

S2SM0：指定串口2的通信工作模式，如下表所示：

S2SM0	串口2工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S2SM2：允许串口 2 在模式 1 时允许多机通信控制位。在模式 1 时，如果 S2SM2 位为 1 且 S2REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 S2RB8）来筛选地址帧：若 S2RB8=1，说明该帧是地址帧，地址信息可以进入 S2BUF，并使 S2RI 为 1，进而在中断服务程序中再进行地址号比较；若 S2RB8=0，说明该帧不是地址帧，应丢掉且保持 S2RI=0。在模式 1 中，如果 S2SM2 位为 0 且 S2REN 位为 1，接收收机处于地址帧筛选被禁止状态。不论收到的 S2RB8 为 0 或 1，均可使接收到的信息进入 S2BUF，并使 S2RI=1，此时 S2RB8 通常为校验位。模式 0 为非多机通信方式，在这种方式时，要设置 S2SM2 应为 0。

S2REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

S2TB8：当串口 2 使用模式 1 时，S2TB8 为要发送的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位，按需要由软件置位或清 0。在模式 0 中，该位不用。

S2RB8：当串口 2 使用模式 1 时，S2RB8 为接收到的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位。在模式 0 中，该位不用。

S2TI：串口 2 发送中断请求标志位。在停止位开始发送时由硬件自动将 S2TI 置 1，向 CPU 发请求中断，响应中断后 S2TI 必须用软件清零。

S2RI：串口 2 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S2RI 置 1，向 CPU 发中断申请，响应中断后 S2RI 必须由软件清零。

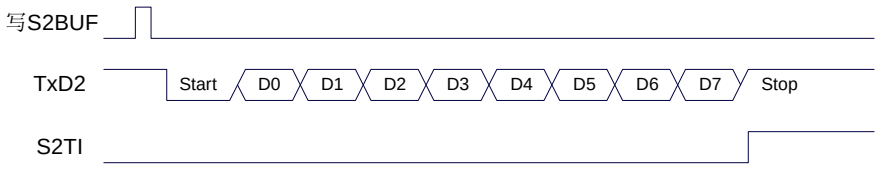
13.3.2 串口 2 数据寄存器（S2BUF）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2BUF	9BH								

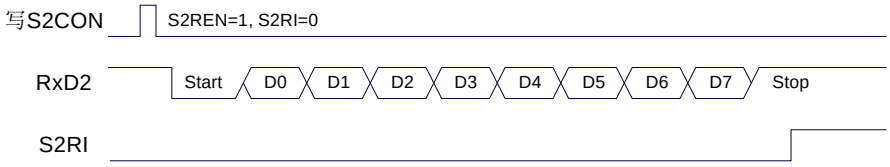
S2BUF：串口 1 数据接收/发送缓冲区。S2BUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 S2BUF 进行读操作，实际是读取串口接收缓冲区，对 S2BUF 进行写操作则是触发串口开始发送数据。

13.3.3 串口 2 模式 0，模式 0 波特率计算公式

串行口 2 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD2 为数据发送口，RxD2 为数据接收口，串行口全双工接受/发送。



发送数据（串口2模式0）



接收数据（串口2模式0）

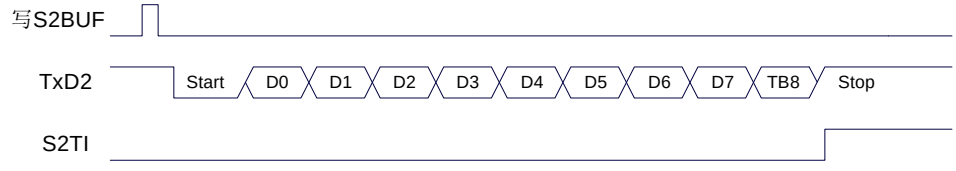
串口 2 的波特率是可变的，其波特率由定时器 2 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 2 模式 0 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

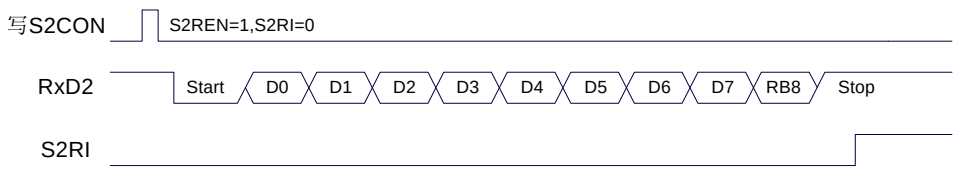
选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536-\frac{SYSclk}{4\times波特率}$
	12T	定时器2重载值 = $65536-\frac{SYSclk}{12\times4\times波特率}$

13.3.4 串口 2 模式 1，模式 1 波特率计算公式

串行口 2 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD2 为数据发送口，RxD2 为数据接收口，串行口全双工接受/发送。



发送数据（串口2模式1）



接收数据（串口2模式1）

串口 2 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

STC MCU

13.4 串口 3

13.4.1 串口 3 控制寄存器（S3CON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3SM0：指定串口3的通信工作模式，如下表所示：

S3SM0	串口3工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S3ST3：选择串口 3 的波特率发生器

0：选择定时器 2 为串口 3 的波特率发生器

1：选择定时器 3 为串口 3 的波特率发生器

S3SM2：允许串口 3 在模式 1 时允许多机通信控制位。在模式 1 时，如果 S3SM2 位为 1 且 S3REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 S3RB8）来筛选地址帧：若 S3RB8=1，说明该帧是地址帧，地址信息可以进入 S3BUF，并使 S3RI 为 1，进而在中断服务程序中再进行地址号比较；若 S3RB8=0，说明该帧不是地址帧，应丢掉且保持 S3RI=0。在模式 1 中，如果 S3SM2 位为 0 且 S3REN 位为 1，接收收机处于地址帧筛选被禁止状态。不论收到的 S3RB8 为 0 或 1，均可使接收到的信息进入 S3BUF，并使 S3RI=1，此时 S3RB8 通常为校验位。模式 0 为非多机通信方式，在这种方式时，要设置 S3SM2 应为 0。

S3REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

S3TB8：当串口 3 使用模式 1 时，S3TB8 为要发送的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位，按需要由软件置位或清 0。在模式 0 中，该位不用。

S3RB8：当串口 3 使用模式 1 时，S3RB8 为接收到的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位。在模式 0 中，该位不用。

S3TI：串口 3 发送中断请求标志位。在停止位开始发送时由硬件自动将 S3TI 置 1，向 CPU 发请求中断，响应中断后 S3TI 必须用软件清零。

S3RI：串口 3 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S3RI 置 1，向 CPU 发中断申请，响应中断后 S3RI 必须由软件清零。

13.4.2 串口 3 数据寄存器（S3BUF）

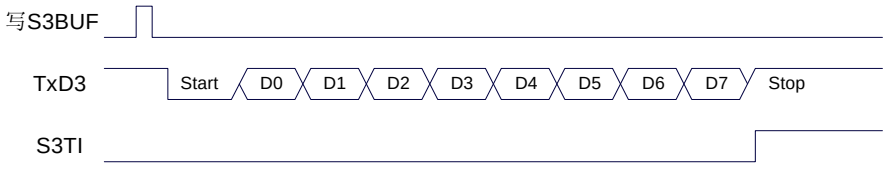
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S3BUF	ADH								

S3BUF：串口 1 数据接收/发送缓冲区。S3BUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 S3BUF 进行读操作，实际是读取串口接收缓冲区，对 S3BUF 进行写操作则是触发串口开始发送数据。

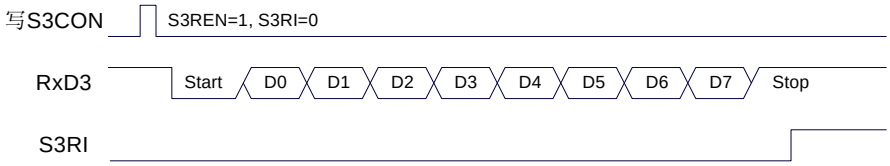
13.4.3 串口 3 模式 0，模式 0 波特率计算公式

串行口 3 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位：1 位起始位，

8 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD3 为数据发送口，RxD3 为数据接收口，串行口全双工接受/发送。



发送数据（串口3模式0）



接收数据（串口3模式0）

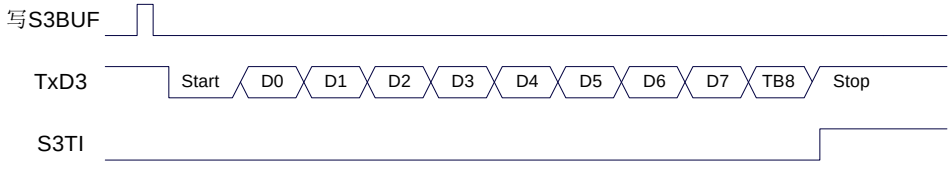
串口 3 的波特率是可变的，其波特率可由定时器 2 或定时器 3 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 3 模式 0 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

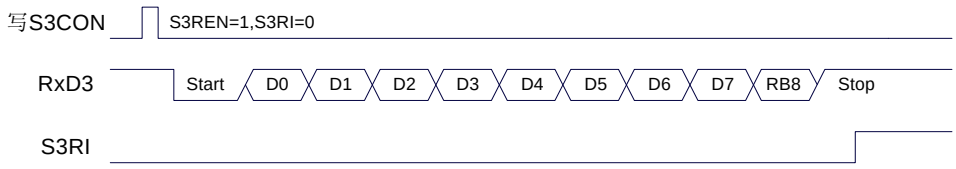
选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536-\frac{SYSclk}{4\times波特率}$
	12T	定时器2重载值 = $65536-\frac{SYSclk}{12\times4\times波特率}$
定时器3	1T	定时器3重载值 = $65536-\frac{SYSclk}{4\times波特率}$
	12T	定时器3重载值 = $65536-\frac{SYSclk}{12\times4\times波特率}$

13.4.4 串口 3 模式 1，模式 1 波特率计算公式

串行口 3 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD3 为数据发送口，RxD3 为数据接收口，串行口全双工接受/发送。



发送数据（串口3模式1）



接收数据（串口3模式1）

串口 3 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

STC MCU

13.5 串口 4

13.5.1 串口 4 控制寄存器（S4CON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4SM0：指定串口4的通信工作模式，如下表所示：

S4SM0	串口4工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S4ST4：选择串口 4 的波特率发生器

0：选择定时器 2 为串口 4 的波特率发生器

1：选择定时器 4 为串口 4 的波特率发生器

S4SM2：允许串口 4 在模式 1 时允许多机通信控制位。在模式 1 时，如果 S4SM2 位为 1 且 S4REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 S4RB8）来筛选地址帧：若 S4RB8=1，说明该帧是地址帧，地址信息可以进入 S4BUF，并使 S4RI 为 1，进而在中断服务程序中再进行地址号比较；若 S4RB8=0，说明该帧不是地址帧，应丢掉且保持 S4RI=0。在模式 1 中，如果 S4SM2 位为 0 且 S4REN 位为 1，接收收机处于地址帧筛选被禁止状态。不论收到的 S4RB8 为 0 或 1，均可使接收到的信息进入 S4BUF，并使 S4RI=1，此时 S4RB8 通常为校验位。模式 0 为非多机通信方式，在这种方式时，要设置 S4SM2 应为 0。

S4REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

S4TB8：当串口 4 使用模式 1 时，S4TB8 为要发送的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位，按需要由软件置位或清 0。在模式 0 中，该位不用。

S4RB8：当串口 4 使用模式 1 时，S4RB8 为接收到的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位。在模式 0 中，该位不用。

S4TI：串口 4 发送中断请求标志位。在停止位开始发送时由硬件自动将 S4TI 置 1，向 CPU 发请求中断，响应中断后 S4TI 必须用软件清零。

S4RI：串口 4 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S4RI 置 1，向 CPU 发中断申请，响应中断后 S4RI 必须由软件清零。

13.5.2 串口 4 数据寄存器（S4BUF）

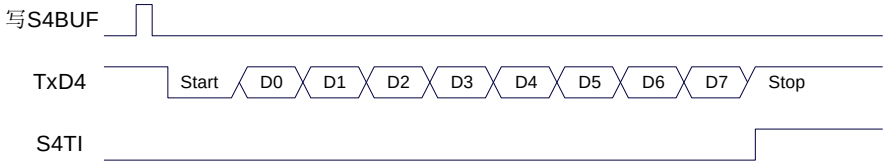
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S4BUF	85H								

S4BUF：串口 1 数据接收/发送缓冲区。S4BUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 S4BUF 进行读操作，实际是读取串口接收缓冲区，对 S4BUF 进行写操作则是触发串口开始发送数据。

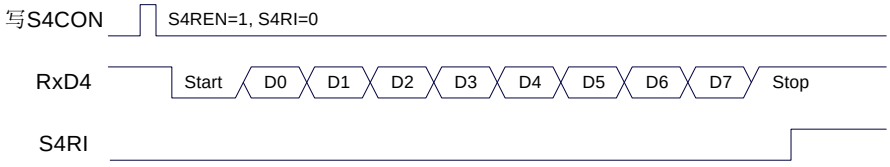
13.5.3 串口 4 模式 0，模式 0 波特率计算公式

串行口 4 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD4 为数据发送口，

RxD4 为数据接收口，串行口全双工接受/发送。



发送数据（串口4模式0）



接收数据（串口4模式0）

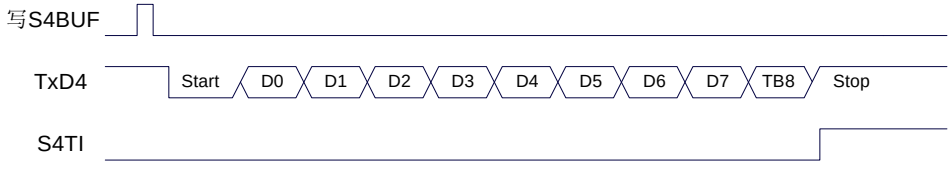
串口 4 的波特率是可变的，其波特率可由定时器 2 或定时器 4 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 4 模式 0 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

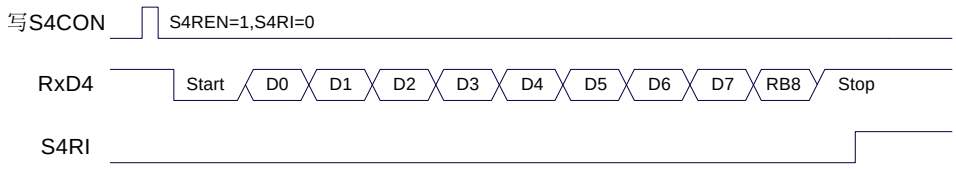
选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536 - \frac{SYSclk}{4\times波特率}$
	12T	定时器2重载值 = $65536 - \frac{SYSclk}{12\times4\times波特率}$
定时器4	1T	定时器4重载值 = $65536 - \frac{SYSclk}{4\times波特率}$
	12T	定时器4重载值 = $65536 - \frac{SYSclk}{12\times4\times波特率}$

13.5.4 串口 4 模式 1，模式 1 波特率计算公式

串行口 4 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD4 为数据发送口，RxD4 为数据接收口，串行口全双工接受/发送。



发送数据（串口4模式1）



接收数据（串口4模式1）

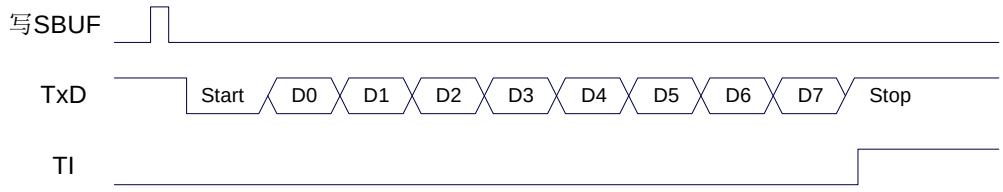
串口 4 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

STC MCU

13.6 串口注意事项

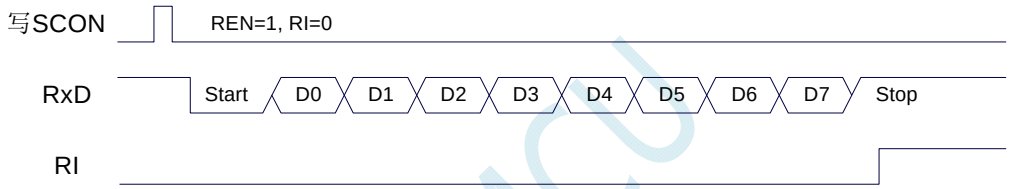
关于串口中断请求有如下问题需要注意：（串口 1、串口 2、串口 3、串口 4 均类似，下面以串口 1 为例进行说明）

8 位数据模式时，发送完成约 1/3 个停止位后产生 TI 中断请求，如下图所示：



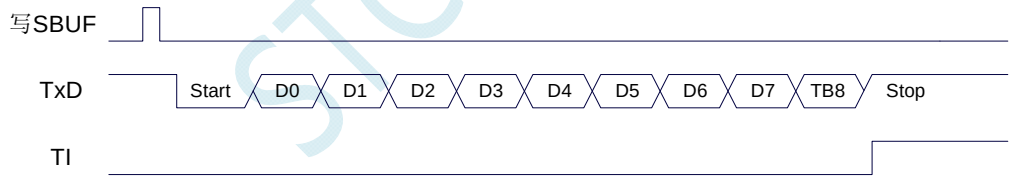
发送数据（8位数据）

8 位数据模式时，接收完成一半个停止位后产生 RI 中断请求，如下图所示：



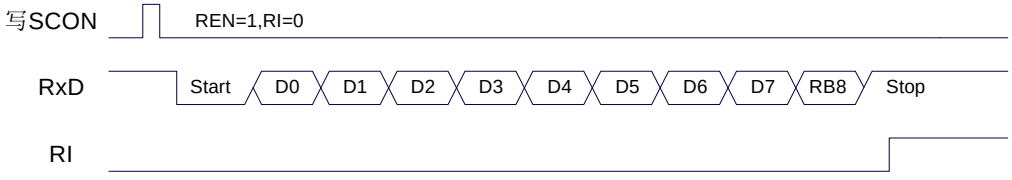
接收数据（8位数据）

9 位数据模式时，发送完成约 1/3 个停止位后产生 TI 中断请求：



发送数据（9位数据）

9 位数据模式时，一半个停止位后产生 RI 中断请求，如下图所示：



接收数据（9位数据）

13.7 范例程序

13.7.1 串口 1 使用定时器 2 做波特率发生器

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;
sfr       T2H         =   0xd6;
sfr       T2L         =   0xd7;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    T2L = BRT;
    T2H = BRT >> 8;
```

STC MCU

```
    AUXR = 0x15;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA        8EH
T2H         DATA        0D6H
T2L         DATA        0D7H
```

```
BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,SBUF
            INC     WPTR
UARTISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART_INIT:
            MOV     SCON,#50H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#15H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART_SEND:
            JB      BUSY,$
```

STC MCU

```
                SETB    BUSY
                MOV     SBUF,A
                RET

UART_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SENDEND
                LCALL   UART_SEND
                INC     DPTR
                JMP     UART_SENDSTR
SENDEND:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

13.7.2 串口 1 使用定时器 1（模式 0）做波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
```

STC MCU

```
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA        8EH

BUSY        BIT         20H.0
WPTR        DATA        21H
RPTR        DATA        22H
BUFFER      DATA        23H                 ;16 bytes

P0M1        DATA        093H
P0M0        DATA        094H
P1M1        DATA        091H
P1M0        DATA        092H
P2M1        DATA        095H
```

```
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,SBUF
            INC     WPTR
UARTISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART_INIT:
            MOV     SCON,#50H
            MOV     TMOD,#00H
            MOV     TL1,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     TH1,#0FFH
            SETB    TR1
            MOV     AUXR,#40H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     SBUF,A
            RET

UART_SENDSTR:
            CLR     A
            MOVC    A,@A+DPTR
            JZ      SENDEND
```

STC MCU

```
                LCALL   UART_SEND
                INC     DPTR
                JMP     UART_SENDSTR
SENDEND:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

13.7.3 串口 1 使用定时器 1（模式 2）做波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define     FOSC            11059200UL
```

```
#define   BRT         (256 - FOSC / 115200 / 32)

sfr       AUXR        =   0x8e;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x20;
    TL1 = BRT;
    TH1 = BRT;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
```

STC MCU

```
    while (*p)
    {
        UartSEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA        8EH

BUSY        BIT         20H.0
WPTR        DATA        21H
RPTR        DATA        22H
BUFFER      DATA        23H                 ;16 bytes

P0M1        DATA        093H
P0M0        DATA        094H
P1M1        DATA        091H
P1M0        DATA        092H
P2M1        DATA        095H
P2M0        DATA        096H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P4M1        DATA        0B3H
P4M0        DATA        0B4H
P5M1        DATA        0C9H
P5M0        DATA        0CAH
```

```
                ORG     0000H
                LJMP    MAIN
                ORG     0023H
                LJMP    UART_ISR

                ORG     0100H

UART_ISR:
                PUSH    ACC
                PUSH    PSW
                MOV     PSW,#08H

                JNB     TI,CHKRI
                CLR     TI
                CLR     BUSY
CHKRI:
                JNB     RI,UARTISR_EXIT
                CLR     RI
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,SBUF
                INC     WPTR
UARTISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#20H
                MOV     TL1,#0FDH       ;256-11059200/115200/32=0FDH
                MOV     TH1,#0FDH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

UART_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     SBUF,A
                RET

UART_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SENDEND
                LCALL   UART_SEND
                INC     DPTR
                JMP     UART_SENDSTR
SENDEND:
                RET

MAIN:
```

STC MCU

```
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                LCALL       UART_INIT
                SETB        ES
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

13.7.4 串口 2 使用定时器 2 做波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC          11059200UL
#define   BRT           (65536 - FOSC / 115200 / 4)

sfr       AUXR          =     0x8e;
sfr       T2H           =     0xd6;
sfr       T2L           =     0xd7;
sfr       S2CON         =     0x9a;
sfr       S2BUF         =     0x9b;
```

```
sfr     IE2     =   0xaf;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

bit     busy;
char    wptr;
char    rptr;
char    buffer[16];

void Uart2Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;
        busy = 0;
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f;
    }
}

void Uart2Init()
{
    S2CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart2Send(char dat)
{
    while (busy);
    busy = 1;
    S2BUF = dat;
}

void Uart2SendStr(char *p)
{
    while (*p)
    {
        Uart2SEND(*p++);
    }
```

STC MCU

```
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart2SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S2CON       DATA    9AH
S2BUF       DATA    9BH
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
```

```
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0043H
            LJMP    UART2_ISR

            ORG     0100H

UART2_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S2CON
            JNB     ACC.1,CHKRI
            ANL     S2CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART2ISR_EXIT
            ANL     S2CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S2BUF
            INC     WPTR
UART2ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART2_INIT:
            MOV     S2CON,#10H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART2_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     S2BUF,A
            RET

UART2_SENDSTR:
            CLR     A
            MOVC    A,@A+DPTR
            JZ      SEND2END
            LCALL   UART2_SEND
            INC     DPTR
            JMP     UART2_SENDSTR
SEND2END:
            RET

MAIN:
```

STC MCU

```
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                LCALL       UART2_INIT
                MOV         IE2,#01H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART2_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART2_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

13.7.5 串口 3 使用定时器 2 做波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC          11059200UL
#define   BRT           (65536 - FOSC / 115200 / 4)

sfr       AUXR          =     0x8e;
sfr       T2H           =     0xd6;
sfr       T2L           =     0xd7;
sfr       S3CON         =     0xac;
sfr       S3BUF         =     0xad;
```

```
sfr     IE2     =   0xaf;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

bit     busy;
char    wptr;
char    rptr;
char    buffer[16];

void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

void Uart3Init()
{
    S3CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3SEND(*p++);
    }
```

STC MCU

```
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
```

```
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3_ISR

            ORG     0100H

UART3_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S3CON
            JNB     ACC.1,CHKRI
            ANL     S3CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART3ISR_EXIT
            ANL     S3CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S3BUF
            INC     WPTR
UART3ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART3_INIT:
            MOV     S3CON,#10H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART3_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     S3BUF,A
            RET

UART3_SENDSTR:
            CLR     A
            MOVC    A,@A+DPTR
            JZ      SEND3END
            LCALL   UART3_SEND
            INC     DPTR
            JMP     UART3_SENDSTR
SEND3END:
            RET

MAIN:
```

STC MCU

```
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART3_INIT
                MOV     IE2,#08H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART3_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART3_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

13.7.6 串口 3 使用定时器 3 做波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       T4T3M   =   0xd1;
sfr       T4L     =   0xd3;
sfr       T4H     =   0xd2;
sfr       T3L     =   0xd5;
sfr       T3H     =   0xd4;
```

```
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     S3CON   =   0xac;
sfr     S3BUF   =   0xad;
sfr     IE2     =   0xaf;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

bit     busy;
char    wptr;
char    rptr;
char    buffer[16];

void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

void Uart3Init()
{
    S3CON = 0x50;
    T3L = BRT;
    T3H = BRT >> 8;
    T4T3M = 0x0a;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
```

STC MCU

```
    while (*p)
    {
        Uart3SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
```

```
P1M0            DATA        092H
P2M1            DATA        095H
P2M0            DATA        096H
P3M1            DATA        0B1H
P3M0            DATA        0B2H
P4M1            DATA        0B3H
P4M0            DATA        0B4H
P5M1            DATA        0C9H
P5M0            DATA        0CAH

                ORG         0000H
                LJMP        MAIN
                ORG         008BH
                LJMP        UART3_ISR

                ORG         0100H

UART3_ISR:
                PUSH        ACC
                PUSH        PSW
                MOV         PSW,#08H

                MOV         A,S3CON
                JNB         ACC.1,CHKRI
                ANL         S3CON,#NOT 02H
                CLR         BUSY
CHKRI:
                JNB         ACC.0,UART3ISR_EXIT
                ANL         S3CON,#NOT 01H
                MOV         A,WPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         @R0,S3BUF
                INC         WPTR
UART3ISR_EXIT:
                POP         PSW
                POP         ACC
                RETI

UART3_INIT:
                MOV         S3CON,#50H
                MOV         T3L,#0E8H           ;65536-11059200/115200/4=0FFE8H
                MOV         T3H,#0FFH
                MOV         T4T3M,#0AH
                CLR         BUSY
                MOV         WPTR,#00H
                MOV         RPTR,#00H
                RET

UART3_SEND:
                JB          BUSY,$
                SETB        BUSY
                MOV         S3BUF,A
                RET

UART3_SENDSTR:
                CLR         A
                MOVC        A,@A+DPTR
```

STC MCU

```
                JZ          SEND3END
                LCALL       UART3_SEND
                INC         DPTR
                JMP         UART3_SENDSTR
SEND3END:
                RET

MAIN:
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                LCALL       UART3_INIT
                MOV         IE2,#08H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART3_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART3_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

13.7.7 串口 4 使用定时器 2 做波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"
```

```
#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;
sfr       T2H         =   0xd6;
sfr       T2L         =   0xd7;
sfr       S4CON       =   0x84;
sfr       S4BUF       =   0x85;
sfr       IE2         =   0xaf;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
```

STC MCU

```
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S4CON       DATA    84H
S4BUF       DATA    85H
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
```

```
P1M0            DATA        092H
P2M1            DATA        095H
P2M0            DATA        096H
P3M1            DATA        0B1H
P3M0            DATA        0B2H
P4M1            DATA        0B3H
P4M0            DATA        0B4H
P5M1            DATA        0C9H
P5M0            DATA        0CAH

                ORG         0000H
                LJMP        MAIN
                ORG         0093H
                LJMP        UART4_ISR

                ORG         0100H

UART4_ISR:
                PUSH        ACC
                PUSH        PSW
                MOV         PSW,#08H

                MOV         A,S4CON
                JNB         ACC.1,CHKRI
                ANL         S4CON,#NOT 02H
                CLR         BUSY
CHKRI:
                JNB         ACC.0,UART4ISR_EXIT
                ANL         S4CON,#NOT 01H
                MOV         A,WPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         @R0,S4BUF
                INC         WPTR
UART4ISR_EXIT:
                POP         PSW
                POP         ACC
                RETI

UART4_INIT:
                MOV         S4CON,#10H
                MOV         T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
                MOV         T2H,#0FFH
                MOV         AUXR,#14H
                CLR         BUSY
                MOV         WPTR,#00H
                MOV         RPTR,#00H
                RET

UART4_SEND:
                JB          BUSY,$
                SETB        BUSY
                MOV         S4BUF,A
                RET

UART4_SENDSTR:
                CLR         A
                MOVC        A,@A+DPTR
```

STC MCU

```
                JZ          SEND4END
                LCALL       UART4_SEND
                INC         DPTR
                JMP         UART4_SENDSTR
SEND4END:
                RET

MAIN:
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                LCALL       UART4_INIT
                MOV         IE2,#10H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART4_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART4_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

13.7.8 串口 4 使用定时器 4 做波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"
```

```
#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       T4T3M       =   0xd1;
sfr       T4L         =   0xd3;
sfr       T4H         =   0xd2;
sfr       T3L         =   0xd5;
sfr       T3H         =   0xd4;
sfr       T2L         =   0xd7;
sfr       T2H         =   0xd6;
sfr       S4CON       =   0x84;
sfr       S4BUF       =   0x85;
sfr       IE2         =   0xaf;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x50;
    T4L = BRT;
    T4H = BRT >> 8;
    T4T3M = 0xa0;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart4Send(char dat)
```

STC MCU

```
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T4T3M   DATA    0D1H
T4L     DATA    0D3H
T4H     DATA    0D2H
T3L     DATA    0D5H
T3H     DATA    0D4H
T2L     DATA    0D7H
T2H     DATA    0D6H
S4CON   DATA    84H
S4BUF   DATA    85H
IE2     DATA    0AFH
```

```
BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0093H
            LJMP    UART4_ISR

            ORG     0100H

UART4_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S4CON
            JNB     ACC.1,CHKRI
            ANL     S4CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART4ISR_EXIT
            ANL     S4CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S4BUF
            INC     WPTR
UART4ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART4_INIT:
            MOV     S4CON,#50H
            MOV     T4L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T4H,#0FFH
            MOV     T4T3M,#0A0H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART4_SEND:
```

```
                JB      BUSY,$
                SETB    BUSY
                MOV     S4BUF,A
                RET

UART4_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SEND4END
                LCALL   UART4_SEND
                INC     DPTR
                JMP     UART4_SENDSTR
SEND4END:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART4_INIT
                MOV     IE2,#10H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART4_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART4_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

13.7.9 串口多机通讯

现参考 STC15 系列数据手册，后续补充

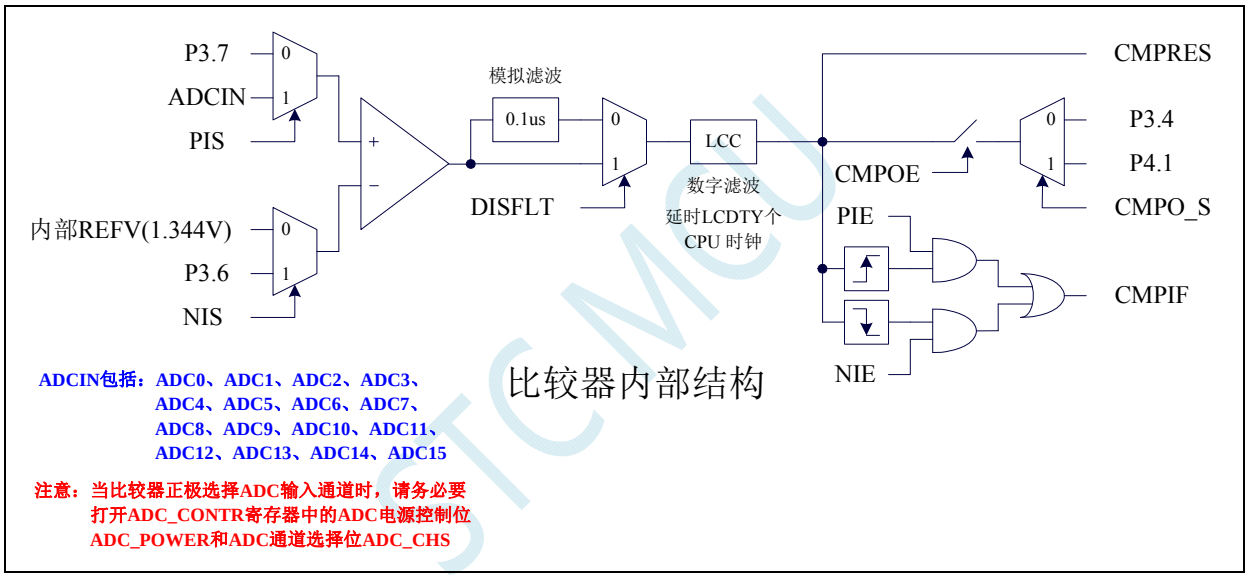
STC MCU

14 比较器，掉电检测，内部 1.344V参考信号源

STC8A/8F 系列单片机内部集成了一个比较器。比较器的正极可以是 P3.7 端口或者 ADC 的模拟输入通道，而负极可以 P3.6 端口或者是内部 BandGap 经过 OP 后的 REFV 电压（内部固定比较电压）。**通过多路选择器和分时复用可实现多个比较器的应用**

比较器内部有可程序控制的两级滤波：模拟滤波和数字滤波。模拟滤波可以过滤掉比较输入信号中的毛刺信号，数字滤波可以等待输入信号更加稳定后再进行比较。比较结果可直接通过读取内部寄存器位获得，也可将比较器结果正向或反向输出到外部端口。将比较结果输出到外部端口可用作外部事件的触发信号和反馈信号，可扩大比较的应用范围。

14.1 比较器内部结构图



比较器内部结构

14.2 比较器相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CMPCR1	比较器控制寄存器 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	比较器控制寄存器 2	E7H	INVCMPO	DISFLT	LCDTY[5:0]						0000,0000

14.2.1 比较器控制寄存器 1（CMPCR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES

CMPEN：比较器模块使能位

0：关闭比较功能

1：使能比较功能

CMPIF：比较器中断标志位。当 PIE 或 NIE 被使能后，若产生相应的中断信号，硬件自动将 CMPIF 置 1，并向 CPU 提出中断请求。此标志位必须用户软件清零。

（注意：没有使能比较器中断时，硬件不会设置此中断标志，即使用查询方式访问比较器时，不能查询此中断标志）

PIE：比较器上升沿中断使能位。

0：禁止比较器上升沿中断。

1：使能比较器上升沿中断。使能比较器的比较结果由 0 变成 1 时产生中断请求。

NIE：比较器下降沿中断使能位。

0：禁止比较器下降沿中断。

1：使能比较器下降沿中断。使能比较器的比较结果由 1 变成 0 时产生中断请求。

PIS：比较器的正极选择位

0：选择外部端口 P3.7 为比较器正极输入源。

1：通过 ADC_CONTR 中的 ADC_CHS 位选择 ADC 的模拟输入端作为比较器正极输入源。

（注意 1：当比较器正极选择 ADC 输入通道时，请务必要打开 ADC_CONTR 寄存器中的 ADC 电源控制位 ADC_POWER 和 ADC 通道选择位 ADC_CHS）

NIS：比较器的负极选择位

0：选择内部 BandGap 经过 OP 后的电压 REFV 作为比较器负极输入源 （芯片在出厂时，内部参考电压调整为 **1.344V**）。

1：选择外部端口 P3.6 为比较器负极输入源。

CMPOE：比较器结果输出控制位

0：禁止比较器结果输出

1：使能比较器结果输出。比较器结果输出到 P3.4 或者 P4.1（由 P_SW2 中的 CMPO_S 进行设定）

CMPRES：比较器的比较结果。此位为只读。

0：表示 CMP+的电平低于 CMP-的电平

1：表示 CMP+的电平高于 CMP-的电平

CMPRES 是经过数字滤波后的输出信号，而不是比较器的直接输出结果。

14.2.2 比较器控制寄存器 2（CMPCR2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR2	E7H	INVCMPO	DISFLT	LCDTY[5:0]					

INVCMPO：比较器结果输出控制

0：比较器结果正向输出。若 CMPRES 为 0，则 P3.4/P4.1 输出低电平，反之输出高电平。

1：比较器结果反向输出。若 CMPRES 为 0，则 P3.4/P4.1 输出高电平，反之输出低电平。

DISFLT：模拟滤波功能控制

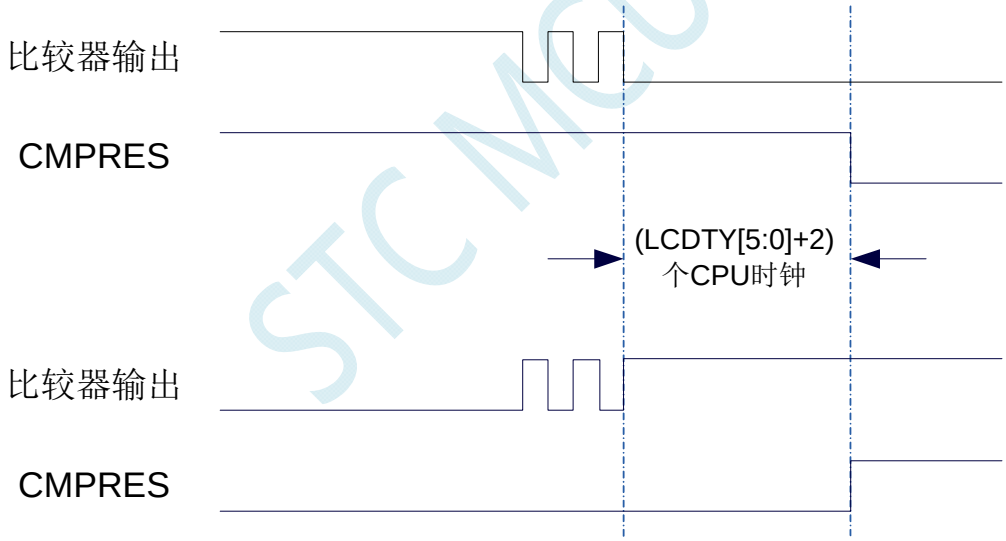
0：使能 0.1us 模拟滤波功能

1：关闭 0.1us 模拟滤波功能，可略微提高比较器的比较速度。

LCDTY[5:0]：数字滤波功能控制

数字滤波功能即为数字信号去抖动功能。当比较结果发生上升沿或者下降沿变化时，比较器侦测变化后的信号必须维持 LCDTY 所设置的 CPU 时钟数不发生变化，才认为数据变化是有效的；否则将视同信号无变化。

注意：当使能数字滤波功能后，芯片内部实际的等待时钟需额外增加两个状态机切换时间，即若 LCDTY 设置为 0 时，为关闭数字滤波功能；若 LCDTY 设置为非 0 值 n（n=1~63）时，则实际的数字滤波时间为（n+2）个系统时钟



STC MCU

14.3 范例程序

14.3.1 比较器的使用（中断方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         CMPCR1          =   0xe6;
sfr         CMPCR2          =   0xe7;

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        P10             =   P1^0;
sbit        P11             =   P1^1;

void CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;                        //清中断标志
    if (CMPCR1 & 0x01)
    {
        P10 = !P10;                         //下降沿中断测试端口
    }
    else
    {
        P11 = !P11;                         //上升沿中断测试端口
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
```

STC MCU

```
    P5M1 = 0x00;

    CMPCR2 = 0x00;
    CMPCR2 &= ~0x80;                    //比较器正向输出
//  CMPCR2 |= 0x80;                     //比较器反向输出
    CMPCR2 &= ~0x40;                    //使能 0.1us 滤波
//  CMPCR2 |= 0x40;                     //禁止 0.1us 滤波
//  CMPCR2 &= ~0x3f;                    //比较器结果直接输出
    CMPCR2 |= 0x10;                     //比较器结果经过 16 个去抖时钟后输出
    CMPCR1 = 0x00;
    CMPCR1 |= 0x30;                     //使能比较器边沿中断
//  CMPCR1 &= ~0x20;                    //禁止比较器上升沿中断
//  CMPCR1 |= 0x20;                     //使能比较器上升沿中断
//  CMPCR1 &= ~0x10;                    //禁止比较器下降沿中断
//  CMPCR1 |= 0x10;                     //使能比较器下降沿中断
    CMPCR1 &= ~0x08;                    //P3.7 为 CMP+输入脚
//  CMPCR1 |= 0x08;                     //ADC 输入脚为 CMP+输入脚
//  CMPCR1 &= ~0x04;                    //内部 1.344V 参考信号源为 CMP-输入脚
    CMPCR1 |= 0x04;                     //P3.6 为 CMP-输入脚
//  CMPCR1 &= ~0x02;                    //禁止比较器输出
    CMPCR1 |= 0x02;                     //使能比较器输出
    CMPCR1 |= 0x80;                     //使能比较器模块

    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00ABH
            LJMP    CMPISR

            ORG     0100H
CMPISR:
            PUSH    ACC
            ANL     CMPCR1,#NOT 40H     ;清中断标志
            MOV     A,CMPCR1
            JB      ACC.0,RSING
```

```
FALLING:
            CPL     P1.0                ;下降沿中断测试端口
            POP     ACC
            RETI
RSING:
            CPL     P1.1                ;上升沿中断测试端口
            POP     ACC
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     CMPCR2,#00H
            ANL     CMPCR2,#NOT 80H     ;比较器正向输出
;           ORL     CMPCR2,#80H         ;比较器反向输出
            ANL     CMPCR2,#NOT 40H     ;使能 0.1us 滤波
;           ORL     CMPCR2,#40H         ;禁止 0.1us 滤波
;           ANL     CMPCR2,#NOT 3FH     ;比较器结果直接输出
            ORL     CMPCR2,#10H         ;比较器结果经过 16 个去抖时钟后输出
            MOV     CMPCR1,#00H
            ORL     CMPCR1,#30H         ;使能比较器边沿中断
;           ANL     CMPCR1,#NOT 20H     ;禁止比较器上升沿中断
;           ORL     CMPCR1,#20H         ;使能比较器上升沿中断
;           ANL     CMPCR1,#NOT 10H     ;禁止比较器下降沿中断
;           ORL     CMPCR1,#10H         ;使能比较器下降沿中断
            ANL     CMPCR1,#NOT 08H     ;P3.7 为CMP+输入脚
;           ORL     CMPCR1,#08H         ;ADC 输入脚为CMP+输入脚
;           ANL     CMPCR1,#NOT 04H     ;内部1.344V 参考信号源为CMP-输入脚
            ORL     CMPCR1,#04H         ;P3.6 为CMP-输入脚
;           ANL     CMPCR1,#NOT 02H     ;禁止比较器输出
            ORL     CMPCR1,#02H         ;使能比较器输出
            ORL     CMPCR1,#80H         ;使能比较器模块
            SETB    EA

            JMP     $

            END
```

14.3.2 比较器的使用（查询方式）

C 语言代码

```
//测试工作频率为11.0592MHz
```

```
#include "reg51.h"
#include "intrins.h"

sfr     CMPCR1      =   0xe6;
sfr     CMPCR2      =   0xe7;

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sbit    P10         =   P1^0;
sbit    P11         =   P1^1;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CMPCR2 = 0x00;
    CMPCR2 &= ~0x80;                    //比较器正向输出
//  CMPCR2 |= 0x80;                     //比较器反向输出
    CMPCR2 &= ~0x40;                    //使能 0.1us 滤波
//  CMPCR2 |= 0x40;                     //禁止 0.1us 滤波
//  CMPCR2 &= ~0x3f;                    //比较器结果直接输出
    CMPCR2 |= 0x10;                     //比较器结果经过 16 个去抖时钟后输出
    CMPCR1 = 0x00;
    CMPCR1 |= 0x30;                     //使能比较器边沿中断
//  CMPCR1 &= ~0x20;                    //禁止比较器上升沿中断
//  CMPCR1 |= 0x20;                     //使能比较器上升沿中断
//  CMPCR1 &= ~0x10;                    //禁止比较器下降沿中断
//  CMPCR1 |= 0x10;                     //使能比较器下降沿中断
    CMPCR1 &= ~0x08;                    //P3.7 为 CMP+输入脚
//  CMPCR1 |= 0x08;                     //ADC 输入脚为 CMP+输入脚
//  CMPCR1 &= ~0x04;                    //内部 1.344V 参考信号源为 CMP-输入脚
    CMPCR1 |= 0x04;                     //P3.6 为 CMP-输入脚
//  CMPCR1 &= ~0x02;                    //禁止比较器输出
    CMPCR1 |= 0x02;                     //使能比较器输出
    CMPCR1 |= 0x80;                     //使能比较器模块

    while (1)
```

STC MCU

```
    {
        P10 = CMPCR1 & 0x01;                    //读取比较器比较结果
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     CMPCR2,#00H
            ANL     CMPCR2,#NOT 80H     ;比较器正向输出
;           ORL     CMPCR2,#80H         ;比较器反向输出
            ANL     CMPCR2,#NOT 40H     ;使能 0.1us 滤波
;           ORL     CMPCR2,#40H         ;禁止 0.1us 滤波
;           ANL     CMPCR2,#NOT 3FH     ;比较器结果直接输出
            ORL     CMPCR2,#10H         ;比较器结果经过 16 个去抖时钟后输出
            MOV     CMPCR1,#00H
            ORL     CMPCR1,#30H         ;使能比较器边沿中断
;           ANL     CMPCR1,#NOT 20H     ;禁止比较器上升沿中断
;           ORL     CMPCR1,#20H         ;使能比较器上升沿中断
;           ANL     CMPCR1,#NOT 10H     ;禁止比较器下降沿中断
;           ORL     CMPCR1,#10H         ;使能比较器下降沿中断
            ANL     CMPCR1,#NOT 08H     ;P3.7 为 CMP+输入脚
;           ORL     CMPCR1,#08H         ;ADC 输入脚为 CMP+输入脚
```

```
;               ANL     CMPCR1,#NOT 04H         ;内部 1.344V 参考信号源为 CMP-输入脚
                ORL     CMPCR1,#04H             ;P3.6 为 CMP-输入脚
;               ANL     CMPCR1,#NOT 02H         ;禁止比较器输出
                ORL     CMPCR1,#02H             ;使能比较器输出
                ORL     CMPCR1,#80H             ;使能比较器模块

LOOP:
                MOV     A,CMPCR1
                MOV     C,ACC.0
                MOV     P1.0,C                  ;读取比较器比较结果
                JMP     LOOP

                END
```

14.3.3 比较器的多路复用应用（比较器+ADC输入通道）

由于比较器的正极可以选择 ADC 的模拟输入通道，因此可以通过多路选择器和分时复用可实现多个比较器的应用。

注意：当比较器正极选择 ADC 输入通道时，请务必要打开 ADC_CONTR 寄存器中的 ADC 电源控制位 ADC_POWER 和 ADC 通道选择位 ADC_CHS

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     CMPCR1  =   0xe6;
sfr     CMPCR2  =   0xe7;

sfr     ADC_CONTR   =   0xbc;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
```

```
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P1M0 &= 0xfe;                   //设置 P1.0 为输入口
    P1M1 |= 0x01;
    ADC_CONTR = 0x80;               //使能 ADC 模块并选择 P1.0 为 ADC 输入脚

    CMPCR2 = 0x00;
    CMPCR1 = 0x00;

    CMPCR1 |= 0x08;                 //ADC 输入脚为 CMP+输入脚
    CMPCR1 |= 0x04;                 //P3.6 为 CMP-输入脚
    CMPCR1 |= 0x02;                 //使能比较器输出
    CMPCR1 |= 0x80;                 //使能比较器模块

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H
ADC_CONTR   DATA    0BDH

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
```

```
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        ANL     P1M0,#0FEH          ;设置 P1.0 为输入口
        ORL     P1M1,#01H
        MOV     ADC_CONTR,#80H      ;使能 ADC 模块并选择 P1.0 为 ADC 输入脚

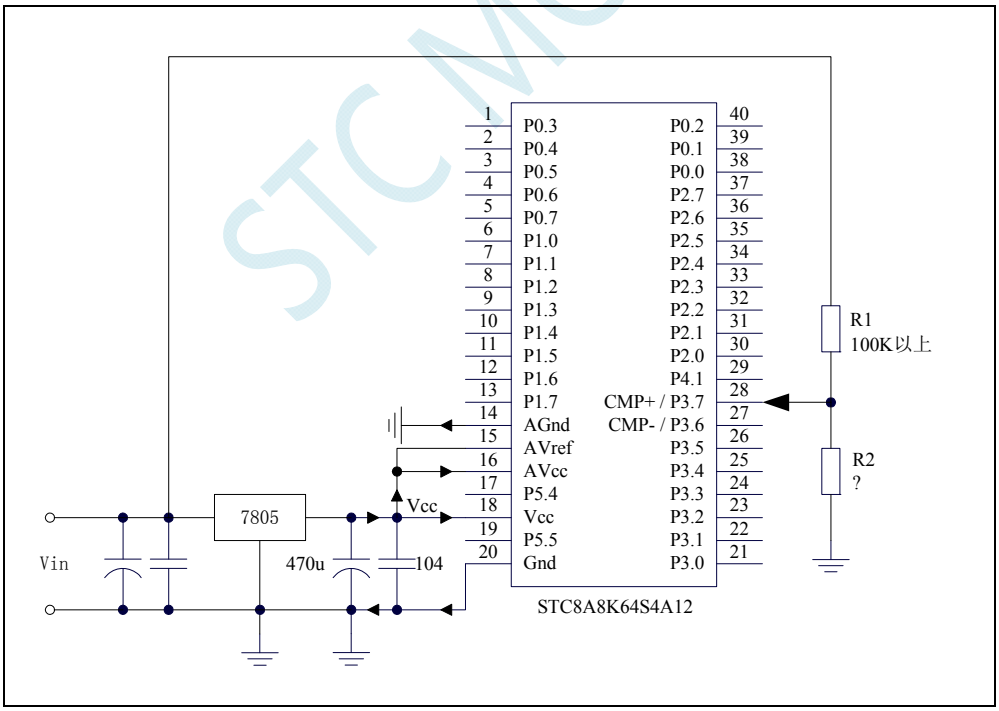
        MOV     CMPCR2,#00H
        MOV     CMPCR1,#00H

        ORL     CMPCR1,#08H         ;ADC 输入脚为 CMP+输入脚
        ORL     CMPCR1,#04H         ;P3.6 为 CMP-输入脚
        ORL     CMPCR1,#02H         ;使能比较器输出
        ORL     CMPCR1,#80H         ;使能比较器模块

LOOP:
        JMP     LOOP

        END
```

14.3.4 比较器作外部掉电检测（掉电过程中应及时保存用户数据到 EEPROM中）



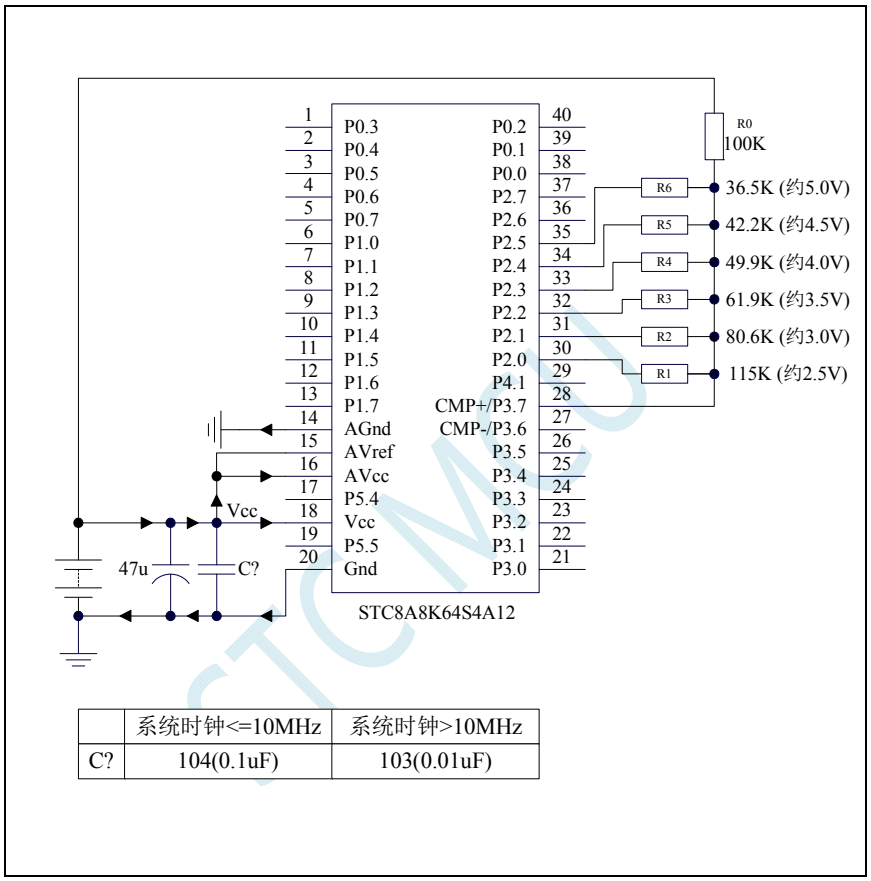
上图中电阻 R1 和 R2 对稳压块 7805 的前端电压进行分压，分压后的电压作为比较器 CMP+的外部输入与内部 1.344V 参考信号源进行比较。

一般当交流电在 220V 时，稳压块 7805 前端的直流电压为 11V，但当交流电压降到 160V 时，稳压块 7805 前端的直留电压为 8.5V。当稳压块 7805 前端的直留电压低于或等于 8.5V 时，该前端输入的直留电压被电阻 R1 和 R2 分压到比较器正极输入端 CMP+，CMP+端输入电压低于内部 1.344V 参考信号源，此时可产生比较器中断，这样在掉电检测时就有充足的时间将数据保存到 EEPROM 中。当稳压块 7805

前端的直留电压高高于8.5V时，该前端输入的直流电压被电阻R1和R2分压到比较器正极输入端CMP+，CMP+端输入电压高于内部1.344V参考信号源，此时CPU可继续正常工作。

内部1.344V参考信号源即为内部BandGap经过OP后的电压REFV（**芯片在出厂时，内部参考电压调整为 1.344V**）。具体的数值要通过读取内部 1.344V参考信号源在内部RAM区或者Flash 程序存储器（ROM）区所占用的地址的值获得。对于STC8系列，内部1.344V参考信号源值在RAM和Flash 程序存储器（ROM）中的存储地址请参考“7.3 章 存储器中的特殊参数”

14.3.5 比较器检测工作电压（电池电压）



	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

上图中，利用电阻分压的原理可以近似的测量出MCU的工作电压（选通的通道，MCU的I/O口输出低电平，端口电压值接近Gnd，未选通的通道，MCU的I/O口输出开漏模式的高，不影响其他通道）。

比较器的负端选择内部1.344V参考信号源，正端选择通过电阻分压后输入到CMP+管脚的电压值。

初始化时P1.5~P1.0口均设置为开漏模式，并输出高。首先P1.0口输出低电平，此时若Vcc电压低于2.5V则比较器的比较值为0，反之若Vcc电压高于2.5V则比较器的比较值为1；

若确定Vcc高于2.5V，则将P1.0口输出高，P1.1口输出低电平，此时若Vcc电压低于3.0V则比较器的比较值为0，反之若Vcc电压高于3.0V则比较器的比较值为1；

若确定Vcc高于3.0V，则将P1.1口输出高，P1.2口输出低电平，此时若Vcc电压低于3.5V则比较器的比较值为0，反之若Vcc电压高于3.5V则比较器的比较值为1；

若确定Vcc高于3.5V，则将P1.2口输出高，P1.3口输出低电平，此时若Vcc电压低于4.0V则比较器的比较值为0，反之若Vcc电压高于4.0V则比较器的比较值为1；

若确定Vcc高于4.0V，则将P1.3口输出高，P1.4口输出低电平，此时若Vcc电压低于4.5V则比较器的比较值为0，反之若Vcc电压高于4.5V则比较器的比较值为1；

若确定Vcc高于4.5V，则将P1.4口输出高，P1.5口输出低电平，此时若Vcc电压低于5.0V则比较

器的比较值为 0，反之若 Vcc 电压高于 5.0V 则比较器的比较值为 1。

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         CMPCR1      =   0xe6;
sfr         CMPCR2      =   0xe7;

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sfr         P2M0        =   0x96;
sfr         P2M1        =   0x95;

void delay ()
{
    char i;

    for (i=0; i<20; i++);
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    unsigned char v;

    P1M0 = 0x3f;                                    //P1.5~P1.0 初始化为开漏模式
    P1M1 = 0x3f;
    P1 = 0xff;

    CMPCR2 = 0x10;                                  //比较器结果经过 16 个去抖时钟后输出
    CMPCR1 = 0x00;
```

```
    CMPCR1 &= ~0x08;                    //P3.7 为CMP+输入脚
    CMPCR1 &= ~0x04;                    //内部1.344V 参考信号源为CMP-输入脚
    CMPCR1 &= ~0x02;                    //禁止比较器输出
    CMPCR1 |= 0x80;                     //使能比较器模块

    while (1)
    {
        v = 0x00;                       //电压<2.5V
        P1 = 0xfe;                      //P1.0 输出0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x01;                       //电压>2.5V
        P1 = 0xfd;                      //P1.1 输出0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x03;                       //电压>3.0V
        P1 = 0xfb;                      //P1.2 输出0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x07;                       //电压>3.5V
        P1 = 0xf7;                      //P1.3 输出0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x0f;                       //电压>4.0V
        P1 = 0xef;                      //P1.4 输出0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x1f;                       //电压>4.5V
        P1 = 0xdf;                      //P1.5 输出0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x3f;                       //电压>5.0V
ShowVol:
        P1 = 0xff;
        P0 = ~v;
    }
}
```

汇编代码

```
;测试工作频率为11.0592MHz

CMPCR1      DATA        0E6H
CMPCR2      DATA        0E7H

P2M0        DATA        096H
P2M1        DATA        095H
P0M1        DATA        093H
P0M0        DATA        094H
P1M1        DATA        091H
P1M0        DATA        092H
P2M1        DATA        095H
P2M0        DATA        096H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P4M1        DATA        0B3H
P4M0        DATA        0B4H
P5M1        DATA        0C9H
P5M0        DATA        0CAH
```

```
        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P1M0,#00111111B     ;P1.5~P1.0 初始化为开漏模式
        MOV     P1M1,#00111111B
        MOV     P1,#0FFH
        MOV     CMPCR2,#10H         ;比较器结果经过 16 个去抖时钟后输出
        MOV     CMPCR1,#00H
        ANL     CMPCR1,#NOT 08H     ;P3.7 为 CMP+输入脚
        ANL     CMPCR1,#NOT 04H     ;内部 1.344V 参考信号源为 CMP-输入脚
        ANL     CMPCR1,#NOT 02H     ;禁止比较器输出
        ORL     CMPCR1,#80H         ;使能比较器模块
LOOP:
        MOV     R0,#00000000B       ;电压<2.5V
        MOV     P1,#11111110B       ;P1.0 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
        JNB     ACC.0,SKIP
        MOV     R0,#00000001B       ;电压>2.5V
        MOV     P1,#11111101B       ;P1.1 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
        JNB     ACC.0,SKIP
        MOV     R0,#00000011B       ;电压>3.0V
        MOV     P1,#11111011B       ;P1.2 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
        JNB     ACC.0,SKIP
        MOV     R0,#00000111B       ;电压>3.5V
        MOV     P1,#11110111B       ;P1.3 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
        JNB     ACC.0,SKIP
        MOV     R0,#00001111B       ;电压>4.0V
        MOV     P1,#11101111B       ;P1.4 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
        JNB     ACC.0,SKIP
        MOV     R0,#00011111B       ;电压>4.5V
        MOV     P1,#11011111B       ;P1.5 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
```

```
            JNB     ACC.0,SKIP
            MOV     R0,#00111111B       ;电压>5.0V
SKIP:
            MOV     P1,#11111111B
            MOV     A,R0
            CPL     A
            MOV     P0,A                ;P0.5~P0.0 口显示电压
            JMP     LOOP

DELAY:
            MOV     R0,#20
            DJNZ    R0,$
            RET

            END
```

STC MCU

15 IAP/EEPROM/DATA-FLASH

STC8A/8F 系列单片机内部集成了大容量的 EEPROM。利用 ISP/IAP 技术可将内部 Data Flash 当 EEPROM，擦写次数在 10 万次以上。EEPROM 可分为若干个扇区，每个扇区包含 512 字节。

注意：EEPROM 的写操作只能将字节中的 1 写为 0，当需要将字节中的 0 写为 1，则必须执行扇区擦除操作。EEPROM 的读/写操作是以 1 字节为单位进行，而 EEPROM 擦除操作是以 1 扇区（512 字节）为单位进行，在执行擦除操作时，如果目标扇区中有需要保留的数据，则必须预先将这些数据读取到 RAM 中暂存，待擦除完成后再将保存的数据和需要更新的数据一起再写回 EEPROM/DATA-FLASH。

所以在使用时，建议同一次修改的数据放在同一个扇区，不是同一次修改的数据放在不同的扇区，不一定要用满。数据存储器的擦除操作是按扇区进行的（每扇区 512 字节）。

EEPROM 可用于保存一些需要在应用过程中修改并且掉电不丢失的参数数据。在用户程序中，可以对 EEPROM 进行字节读/字节编程/扇区擦除操作。在工作电压偏低时，建议不要进行 EEPROM 操作，以免发送数据丢失的情况。

15.1 EEPROM操作时间

- 读取 1 字节：4 个系统时钟（使用 MOVC 指令读取更方便快捷）
- 编程 1 字节：约 30～40us（实际的编程时间为 6~7.5us，但还需要加上状态转换时间和各种控制信号的 SETUP 和 HOLD 时间）
- 擦除 1 扇区（512 字节）：约 4～6ms

15.2 EEPROM相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IAP_DATA	IAP 数据寄存器	C2H									1111,1111
IAP_ADDRH	IAP 高地址寄存器	C3H									0000,0000
IAP_ADDRL	IAP 低地址寄存器	C4H									0000,0000
IAP_CMD	IAP 命令寄存器	C5H	-	-	-	-	-	-	CMD[1:0]		xxxx,xx00
IAP_TRIG	IAP 触发寄存器	C6H									0000,0000
IAP_CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	IAP_WT[2:0]			0000,x000

15.2.1 EEPROM数据寄存器（IAP_DATA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_DATA	C2H								

在进行 EEPROM 的读操作时，命令执行完成后读出的 EEPROM 数据保存在 IAP_DATA 寄存器中。在进行 EEPROM 的写操作时，在执行写命令前，必须将待写入的数据存放在 IAP_DATA 寄存器中，再发送写命令。擦除 EEPROM 命令与 IAP_DATA 寄存器无关。

15.2.2 EEPROM地址寄存器（IAP_ADDR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_ADDRH	C3H								
IAP_ADDRL	C4H								

EEPROM 进行读、写、擦除操作的目标地址寄存器。IAP_ADDRH 保存地址的高字节，IAP_ADDRL 保存地址的低字节

15.2.3 EEPROM命令寄存器（IAP_CMD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CMD	C5H	-	-	-	-	-	-	CMD[1:0]	

CMD[1:0]：发送EEPROM操作命令

00：空操作

01：读 EEPROM 命令。读取目标地址所在的 1 字节。

10：写 EEPROM 命令。写目标地址所在的 1 字节。注意：写操作只能将目标字节中的 1 写为 0，而不能将 0 写为 1。一般当目标字节不为 FFH 时，必须先擦除。

11：擦除 EEPROM。擦除目标地址所在的 1 页（1 扇区/512 字节）。注意：擦除操作会一次擦除 1 个扇区（512 字节），整个扇区的内容全部变成 FFH。

15.2.4 EEPROM触发寄存器（IAP_TRIG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TRIG	C6H								

设置完成 EEPROM 读、写、擦除的命令寄存器、地址寄存器、数据寄存器以及控制寄存器后，需要向触发寄存器 IAP_TRIG 依次写入 5AH、A5H（顺序不能交换）两个触发命令来触发相应的读、写、擦除操作。操作完成后，EEPROM 地址寄存器 IAP_ADDRH、IAP_ADDRL 和 EEPROM 命令寄存器 IAP_CMD 的内容不变。如果接下来要对下一个地址的数据进行操作，需手动更新地址寄存器 IAP_ADDRH 和寄存器 IAP_ADDRL 的值。

注意：每次 EEPROM 操作时，都要对 IAP_TRIG 先写入 5AH，再写入 A5H，相应的命令才会生效。写完触发命令后，CPU 会处于 IDLE 等待状态，直到相应的 IAP 操作执行完成后 CPU 才会从 IDLE 状态返回正常状态继续执行 CPU 指令。

15.2.5 EEPROM控制寄存器（IAP_CONTR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	IAP_WT[2:0]		

IAPEN：EEPROM操作使能控制位

0：禁止 EEPROM 操作

1：使能 EEPROM 操作

SWBS：软件复位选择控制位，（需要与SWRST配合使用）

0：软件复位后从用户代码开始执行程序

1：软件复位后从系统 ISP 监控代码区开始执行程序

SWRST：软件复位控制位

0：无动作

1：产生软件复位

CMD_FAIL：EEPROM操作失败状态位，需要软件清零

0：EEPROM 操作正确

1：EEPROM 操作失败

IAP_WT[2:0]：设置EEPROM操作的等待时间

IAP_WT[2:0]			读字节 （2 个时钟）	写字节 （约 7us）	擦除扇区 （约 5ms）	时钟频率
1	1	1	2 个时钟	7 个时钟	5000 个时钟	**1MHz**
1	1	0	2 个时钟	14 个时钟	10000 个时钟	**2MHz**
1	0	1	2 个时钟	21 个时钟	15000 个时钟	**3MHz**
1	0	0	2 个时钟	42 个时钟	30000 个时钟	**6MHz**
0	1	1	2 个时钟	84 个时钟	60000 个时钟	**12MHz**
0	1	0	2 个时钟	140 个时钟	100000 个时钟	**20MHz**
0	0	1	2 个时钟	168 个时钟	120000 个时钟	**24MHz**
0	0	0	2 个时钟	301 个时钟	215000 个时钟	**30MHz**

此时 MCU 系统不给 CPU 供应时钟，CPU 没有时钟，所以无法工作，也就是说，针对 EEPROM 操作所需要的等待时间是硬件自动完成的，用户不需要加额外的软件延时。但 MCU 给串口、定时器、SPI、I2C 等外设供应时钟，故除 CPU 无法执行指令外，其他外设仍然继续工作。

EEPROM的读操作其实可不用IAP读方式，可用MOVC指令进行读取，CPU就可继续执行指令，不用等待两个时钟。（汇编、C如何读待后续介绍）

15.3 关于EEPROM编程和擦除等待时间的重要说明

表一（STC8A 系列和 STC8F 系列 EEPROM 操作时间需求）

EEPROM 操作	最短时间	最长时间
编程	6us	7.5us
擦除	4ms	6ms

表二（STC8A 系列和 STC8F 系列 EEPROM 操作相应等待参数的时间等待周期）

IAP_WT[2:0]			编程等待时钟数	擦除等待时钟数	适合的频率
1	1	1	7 个时钟	5000 个时钟	1MHz
1	1	0	14 个时钟	10000 个时钟	2MHz
1	0	1	21 个时钟	15000 个时钟	3MHz
1	0	0	42 个时钟	30000 个时钟	6MHz
0	1	1	84 个时钟	60000 个时钟	12MHz
0	1	0	140 个时钟	100000 个时钟	20MHz
0	0	1	168 个时钟	120000 个时钟	24MHz
0	0	0	301 个时钟	215000 个时钟	30MHz

STC8A 系列和 STC8F 系列 MCU 的内部 EEPROM 的编程和擦除等待时间必须达到表一中的要求，等待时间不可过短，也不可过长。

编程的等待时间必须在 6us～7.5us 之间，编程等待时间过小（小于最短时间 6us），则被编程的目标存储单元内部的数据可能不可靠（数据的保存期限可能达不到 25 年）；若等待时间过长（大于最长时间 7.5us 的 1.5 倍，即大于 11.25us），也可能由于有数据干扰而导致写入的数据不正确。在确保编程的等待时间要求，并在编程完成后进行数据读出对比校验，若校验正确，数据便编程正确了。

擦除的等待时间必须在 4ms～6ms 之间，擦除等待时间过小（小于最短时间 4ms），则被擦除的目标存储扇区可能没有被擦除干净；若等待时间过长（大于最长时间 6ms 的 1.5 倍，即大于 9ms），则会缩短 EEPROM 的使用寿命，即原本 10 万次的擦除寿命可能会缩短为 5 万次。

编程与擦除的等待时间请严格按照表二所给的推荐频率进行合适的选择，假如工作频率为 12MHz，请按照表二推荐将等待参数设置为 011B，若 CPU 实际的工作频率并不在表二所推荐的频率之列，则需要根据实际的频率以及表二中实际的等待时钟数进行计算，找出满足表一时间需求的等待时间参数。

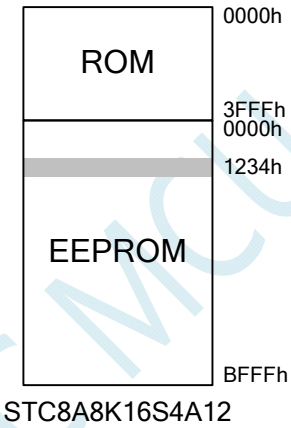
例如：工作频率为 4MHz，若选择等待参数为 101B，则编程时间为 21/4MHz = 5.25us，擦除时间为 15000/4MHz = 3.75ms，时间明显不够，所以应该选择等待参数为 100B，则编程时间为 42/4MHz = 10.5us，擦除时间为 30000/4MHz = 7.5ms，时间均在最短时间和最长时间的 1.5 倍之间。

注意：EEPROM 等待操作的时钟是指对主时钟进行分频后的系统时钟，即 CPU 实际的工作时钟。若单片机使用的是内部高精度 IRC，则 EEPROM 等待操作的时钟为使用 ISP 下载软件下载时经过调节后的频率；若单片机使用的外部晶振，则 EEPROM 等待操作的时钟为外部晶振频率经过 CLKDIV 寄存器分频后的时钟（例如：若单片机使用外部晶振，且外部晶振的频率为 24MHz，CLKDIV 寄存器的值设置为 4，则 EEPROM 等待操作的时钟频率为 24MHz/4 = 6MHz，此时等待参数应选择 100B，而不能选择 001B）。

15.4 EEPROM大小及地址

STC8A/8F 系列单片机内部均有用于保存用户数据的 EEPROM。内部的 EEPROM 有 3 操作方式：读、写和擦除，其中擦除操作是以扇区为单位进行操作，每扇区为 512 字节，即每执行一次擦除命令就会擦除一个扇区，而读数据和写数据都是以字节为单位进行操作的，即每执行一次读或者写命令时只能读出或者写入一个字节。

STC8A/8F 系列单片机内部的 EEPROM 的访问方式有两种：IAP 方式和 MOVC 方式。IAP 方式可对 EEPROM 执行读、写、擦除操作，但 MOVC 只能对 EEPROM 进行读操作，而不能进行写和擦除操作。无论是使用 IAP 方式还是使用 MOVC 方式访问 EEPROM，首先都需要设置正确的目标地址。IAP 方式时，目标地址与 EEPROM 实际的物理地址是一致的，均是从地址 0000H 开始访问，但若要使用 MOVC 指令进行读取 EEPROM 数据时，目标地址必须是在 EEPROM 实际的物理地址的基础上还有加上程序大小的偏移。下面以 STC8A8K16S4A12 这个型号为例，对目标地址进行详细说明：



STC8A8K16S4A12

STC8A8K16S4A12 的程序空间为 16K 字节（0000h~3FFFh），EEPROM 空间为 48K（0000h~BFFFh）。当需要对 EEPROM 物理地址 1234h 的单元进行读、写、擦除时，若使用 IAP 方式进行访问时，设置的目标地址为 1234h，即 IAP_ADDRH 设置 12h，IAP_ADDRL 设置 34h，然后设置相应的触发命令即可对 1234h 单元进行正确操作了。但若是使用 MOVC 方式读取 EEPROM 的 1234h 单元，则必须在 1234h 的基础上还有加上 ROM 空间的大小 4000h，即必须将 DPTR 设置为 5234h，然后才能使用 MOVC 指令进行读取。

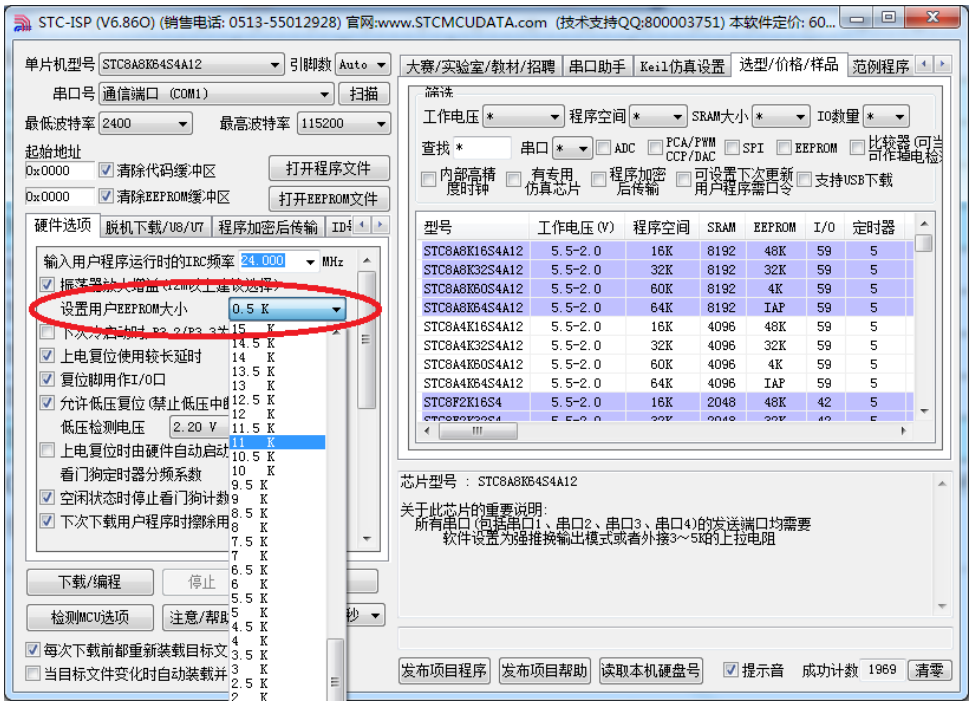
注意：由于擦除是以 512 字节为单位进行操作的，所以执行擦除操作时所设置的目标地址的低 9 位是无意义的。例如：执行擦除命令时，设置地址 0234H/0200H/0300H/03FFH，最终执行擦除的动作都是相同的，都是擦除 0200H~03FFH 这 512 字节。

不同型号内部 EEPROM 的大小及访问地址会存在差异，针对各个型号 EEPROM 的详细大小和地址请参考下表

型号	大小	扇区	IAP 方式读/写/擦除		MOVC 读取	
			起始地址	结束地址	起始地址	结束地址
STC8A8K08S4A12	56K	112	0000h	DFFFh	2000h	FFFFh
STC8A8K16S4A12	48K	96	0000h	BFFFh	4000h	FFFFh
STC8A8K24S4A12	40K	80	0000h	9FFFh	6000h	FFFFh
STC8A8K32S4A12	32K	64	0000h	7FFFh	8000h	FFFFh

STC8A8K40S4A12	24K	48	0000h	5FFFh	A000h	FFFFh
STC8A8K48S4A12	16K	32	0000h	3FFFh	C000h	FFFFh
STC8A8K56S4A12	8K	16	0000h	1FFFh	E000h	FFFFh
STC8A8K60S4A12	4K	8	0000h	0FFFh	F000h	FFFFh
STC8A8K64S4A12	用户自定义[1]					
STC8A4K08S2A12	56K	112	0000h	DFFFh	2000h	FFFFh
STC8A4K16S2A12	48K	96	0000h	BFFFh	4000h	FFFFh
STC8A4K24S2A12	40K	80	0000h	9FFFh	6000h	FFFFh
STC8A4K32S2A12	32K	64	0000h	7FFFh	8000h	FFFFh
STC8A4K40S2A12	24K	48	0000h	5FFFh	A000h	FFFFh
STC8A4K48S2A12	16K	32	0000h	3FFFh	C000h	FFFFh
STC8A4K56S2A12	8K	16	0000h	1FFFh	E000h	FFFFh
STC8A4K60S2A12	4K	8	0000h	0FFFh	F000h	FFFFh
STC8A4K64S2A12	用户自定义[1]					
STC8F2K08S4	56K	112	0000h	DFFFh	2000h	FFFFh
STC8F2K16S4	48K	96	0000h	BFFFh	4000h	FFFFh
STC8F2K24S4	40K	80	0000h	9FFFh	6000h	FFFFh
STC8F2K32S4	32K	64	0000h	7FFFh	8000h	FFFFh
STC8F2K40S4	24K	48	0000h	5FFFh	A000h	FFFFh
STC8F2K48S4	16K	32	0000h	3FFFh	C000h	FFFFh
STC8F2K56S4	8K	16	0000h	1FFFh	E000h	FFFFh
STC8F2K60S4	4K	8	0000h	0FFFh	F000h	FFFFh
STC8F2K64S4	用户自定义[1]					
STC8F2K08S2	56K	112	0000h	DFFFh	2000h	FFFFh
STC8F2K16S2	48K	96	0000h	BFFFh	4000h	FFFFh
STC8F2K24S2	40K	80	0000h	9FFFh	6000h	FFFFh
STC8F2K32S2	32K	64	0000h	7FFFh	8000h	FFFFh
STC8F2K40S2	24K	48	0000h	5FFFh	A000h	FFFFh
STC8F2K48S2	16K	32	0000h	3FFFh	C000h	FFFFh
STC8F2K56S2	8K	16	0000h	1FFFh	E000h	FFFFh
STC8F2K60S2	4K	8	0000h	0FFFh	F000h	FFFFh
STC8F2K64S2	用户自定义[1]					
STC8F1K02S2	10K	20	0000h	27FFh	0800h	2FFFh
STC8F1K04S2	8K	16	0000h	1FFFh	1000h	2FFFh
STC8F1K06S2	6K	12	0000h	17FFh	1800h	2FFFh
STC8F1K08S2	4K	8	0000h	0FFFh	2000h	2FFFh
STC8F1K10S2	2K	4	0000h	07FFh	2800h	2FFFh
STC8F1K12S2	用户自定义[1]					
STC8F1K17S2	用户自定义[1]					

[1]：这个为特殊型号，这个型号的 EEPROM 大小是可用在 ISP 下载时用户自己设置的。如下图所示：



用户可用根据自己的需要在整个 FLASH 空间中规划出任意不超过 FLASH 大小的 EEPROM 空间，但需要注意：**EEPROM 总是从后向前进行规划的**。

例如：STC8A8K64S4A12 这个型号的 FLASH 为 64K，此时若用户想分出其中的 8K 作为 EEPROM 使用，则 EEPROM 的物理地址则为 64K 的最后 8K，物理地址为 E000h~FFFFh，当然，用户若使用 IAP 的方式进行访问，目标地址仍然从 0000h 开始，到 1FFFh 结束，当使用 MOVC 读取则需要从 E000h 开始，到 FFFFh 结束。

15.5 范例程序

15.5.1 EEPROM基本操作

汇编代码

```
;测试工作频率为11.0592MHz

IAP_DATA    DATA    0C2H
IAP_ADDRH   DATA    0C3H
IAP_ADDRL   DATA    0C4H
IAP_CMD     DATA    0C5H
IAP_TRIG    DATA    0C6H
IAP_CONTR   DATA    0C7H

WT_30M      EQU     80H
WT_24M      EQU     81H
WT_20M      EQU     82H
WT_12M      EQU     83H
WT_6M       EQU     84H
WT_3M       EQU     85H
WT_2M       EQU     86H
WT_1M       EQU     87H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H

IAP_IDLE:
            MOV     IAP_CONTR,#0            ;关闭IAP功能
            MOV     IAP_CMD,#0              ;清除命令寄存器
            MOV     IAP_TRIG,#0             ;清除触发寄存器
            MOV     IAP_ADDRH,#80H          ;将地址设置到非IAP区域
            MOV     IAP_ADDRL,#0
            RET

IAP_READ:
            MOV     IAP_CONTR,#WT_12M       ;使能IAP
            MOV     IAP_CMD,#1              ;设置IAP读命令
            MOV     IAP_ADDRL,DPL           ;设置IAP低地址
            MOV     IAP_ADDRH,DPH           ;设置IAP高地址
            MOV     IAP_TRIG,#5AH           ;写触发命令(0x5a)
            MOV     IAP_TRIG,#0A5H          ;写触发命令(0xa5)
            NOP
            MOV     A,IAP_DATA              ;读取IAP数据
            LCALL   IAP_IDLE                ;关闭IAP功能
            RET

IAP_PROGRAM:
            MOV     IAP_CONTR,#WT_12M       ;使能IAP
            MOV     IAP_CMD,#2              ;设置IAP写命令
            MOV     IAP_ADDRL,DPL           ;设置IAP低地址
            MOV     IAP_ADDRH,DPH           ;设置IAP高地址
            MOV     IAP_DATA,A              ;写IAP数据
            MOV     IAP_TRIG,#5AH           ;写触发命令(0x5a)
            MOV     IAP_TRIG,#0A5H          ;写触发命令(0xa5)
```

```
                NOP
                LCALL   IAP_IDLE                ;关闭 IAP 功能
                RET

IAP_ERASE:
                MOV     IAP_CONTR,#WT_12M       ;使能 IAP
                MOV     IAP_CMD,#3              ;设置 IAP 擦除命令
                MOV     IAP_ADDRL,DPL           ;设置 IAP 低地址
                MOV     IAP_ADDRH,DPH           ;设置 IAP 高地址
                MOV     IAP_TRIG,#5AH           ;写触发命令(0x5a)
                MOV     IAP_TRIG,#0A5H          ;写触发命令(0xa5)
                NOP
                LCALL   IAP_IDLE                ;关闭 IAP 功能
                RET

MAIN:
                MOV     SP,#3FH

                MOV     DPTR,#0400H
                LCALL   IAP_ERASE
                MOV     DPTR,#0400H
                LCALL   IAP_READ
                MOV     P0,A                    ;P0=0FFH
                MOV     DPTR,#0400H
                MOV     A,#12H
                LCALL   IAP_PROGRAM
                MOV     DPTR,#0400H
                LCALL   IAP_READ
                MOV     P1,A                    ;P1=12H

                SJMP    $

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为 11.0592MHz

sfr     IAP_DATA    =   0xC2;
sfr     IAP_ADDRH   =   0xC3;
sfr     IAP_ADDRL   =   0xC4;
sfr     IAP_CMD     =   0xC5;
sfr     IAP_TRIG    =   0xC6;
sfr     IAP_CONTR   =   0xC7;

#define WT_30M          0x80
#define WT_24M          0x81
#define WT_20M          0x82
#define WT_12M          0x83
#define WT_6M           0x84
#define WT_3M           0x85
#define WT_2M           0x86
#define WT_1M           0x87

void IapIdle()
{
```

```
        IAP_CONTR = 0;              //关闭 IAP 功能
        IAP_CMD = 0;                //清除命令寄存器
        IAP_TRIG = 0;               //清除触发寄存器
        IAP_ADDRH = 0x80;           //将地址设置到非 IAP 区域
        IAP_ADDRL = 0;
}

char IapRead(int addr)
{
        char dat;

        IAP_CONTR = WT_12M;         //使能 IAP
        IAP_CMD = 1;                //设置 IAP 读命令
        IAP_ADDRL = addr;           //设置 IAP 低地址
        IAP_ADDRH = addr >> 8;      //设置 IAP 高地址
        IAP_TRIG = 0x5a;            //写触发命令(0x5a)
        IAP_TRIG = 0xa5;            //写触发命令(0xa5)
        _nop_();
        dat = IAP_DATA;             //读 IAP 数据
        IapIdle();                  //关闭 IAP 功能

        return dat;
}

void IapProgram(int addr, char dat)
{
        IAP_CONTR = WT_12M;         //使能 IAP
        IAP_CMD = 2;                //设置 IAP 写命令
        IAP_ADDRL = addr;           //设置 IAP 低地址
        IAP_ADDRH = addr >> 8;      //设置 IAP 高地址
        IAP_DATA = dat;             //写 IAP 数据
        IAP_TRIG = 0x5a;            //写触发命令(0x5a)
        IAP_TRIG = 0xa5;            //写触发命令(0xa5)
        _nop_();
        IapIdle();                  //关闭 IAP 功能
}

void IapErase(int addr)
{
        IAP_CONTR = WT_12M;         //使能 IAP
        IAP_CMD = 3;                //设置 IAP 擦除命令
        IAP_ADDRL = addr;           //设置 IAP 低地址
        IAP_ADDRH = addr >> 8;      //设置 IAP 高地址
        IAP_TRIG = 0x5a;            //写触发命令(0x5a)
        IAP_TRIG = 0xa5;            //写触发命令(0xa5)
        _nop_();                    //
        IapIdle();                  //关闭 IAP 功能
}

void main()
{
        IapErase(0x0400);
        P0 = IapRead(0x0400);       //P0=0xff
        IapProgram(0x0400, 0x12);
        P1 = IapRead(0x0400);       //P1=0x12

        while (1);
}
```

STCMCU

15.5.2 使用MOVC读取EEPROM

汇编代码

```
;测试工作频率为11.0592MHz

IAP_DATA    DATA    0C2H
IAP_ADDRH   DATA    0C3H
IAP_ADDRL   DATA    0C4H
IAP_CMD     DATA    0C5H
IAP_TRIG    DATA    0C6H
IAP_CONTR   DATA    0C7H

WT_30M      EQU     80H
WT_24M      EQU     81H
WT_20M      EQU     82H
WT_12M      EQU     83H
WT_6M       EQU     84H
WT_3M       EQU     85H
WT_2M       EQU     86H
WT_1M       EQU     87H

IAP_OFFSET  EQU     2000H           ;STC8A8K08S4A12
;IAP_OFFSET EQU     4000H           ;STC8A8K16S4A12
;IAP_OFFSET EQU     6000H           ;STC8A8K24S4A12
;IAP_OFFSET EQU     8000H           ;STC8A8K32S4A12
;IAP_OFFSET EQU     0A000H          ;STC8A8K40S4A12
;IAP_OFFSET EQU     0C000H          ;STC8A8K48S4A12
;IAP_OFFSET EQU     0E000H          ;STC8A8K56S4A12
;IAP_OFFSET EQU     0F000H          ;STC8A8K60S4A12

            ORG     0000H
            LJMP    MAIN

            ORG     0100H

IAP_IDLE:
            MOV     IAP_CONTR,#0        ;关闭IAP 功能
            MOV     IAP_CMD,#0          ;清除命令寄存器
            MOV     IAP_TRIG,#0         ;清除触发寄存器
            MOV     IAP_ADDRH,#80H      ;将地址设置到非IAP 区域
            MOV     IAP_ADDRL,#0
            RET

IAP_READ:
            MOV     A,#LOW IAP_OFFSET   ;使用MOVC 读取EEPROM 需要加上相应的偏移
            ADD     A,DPL
            MOV     DPL,A
            MOV     A,@HIGH IAP_OFFSET
            ADDC    A,DPH
            MOV     DPH,A
            CLR     A
            MOVC    A,@A+DPTR           ;使用MOVC 读取数据
            RET

IAP_PROGRAM:
            MOV     IAP_CONTR,#WT_12M   ;使能IAP
            MOV     IAP_CMD,#2          ;设置IAP 写命令
            MOV     IAP_ADDRL,DPL       ;设置IAP 低地址
```

```
                MOV     IAP_ADDRH,DPH           ;设置 IAP 高地址
                MOV     IAP_DATA,A              ;写 IAP 数据
                MOV     IAP_TRIG,#5AH           ;写触发命令(0x5a)
                MOV     IAP_TRIG,#0A5H          ;写触发命令(0xa5)
                NOP
                LCALL   IAP_IDLE                ;关闭 IAP 功能
                RET

IAP_ERASE:
                MOV     IAP_CONTR,#WT_12M       ;使能 IAP
                MOV     IAP_CMD,#3              ;设置 IAP 擦除命令
                MOV     IAP_ADDRL,DPL           ;设置 IAP 低地址
                MOV     IAP_ADDRH,DPH           ;设置 IAP 高地址
                MOV     IAP_TRIG,#5AH           ;写触发命令(0x5a)
                MOV     IAP_TRIG,#0A5H          ;写触发命令(0xa5)
                NOP
                LCALL   IAP_IDLE                ;关闭 IAP 功能
                RET

MAIN:
                MOV     SP,#3FH

                MOV     DPTR,#0400H
                LCALL   IAP_ERASE
                MOV     DPTR,#0400H
                LCALL   IAP_READ
                MOV     P0,A                    ;P0=0FFH
                MOV     DPTR,#0400H
                MOV     A,#12H
                LCALL   IAP_PROGRAM
                MOV     DPTR,#0400H
                LCALL   IAP_READ
                MOV     P1,A                    ;P1=12H

                SJMP    $

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为 11.0592MHz

sfr     IAP_DATA    =   0xC2;
sfr     IAP_ADDRH   =   0xC3;
sfr     IAP_ADDRL   =   0xC4;
sfr     IAP_CMD     =   0xC5;
sfr     IAP_TRIG    =   0xC6;
sfr     IAP_CONTR   =   0xC7;

#define WT_30M          0x80
#define WT_24M          0x81
#define WT_20M          0x82
#define WT_12M          0x83
#define WT_6M           0x84
#define WT_3M           0x85
#define WT_2M           0x86
```

```
#define    WT_1M           0x87

#define    IAP_OFFSET      0x2000H         //STC8A8K08S4A12
//#define  IAP_OFFSET      0x4000H         //STC8A8K16S4A12
//#define  IAP_OFFSET      0x6000H         //STC8A8K24S4A12
//#define  IAP_OFFSET      0x8000H         //STC8A8K32S4A12
//#define  IAP_OFFSET      0xA000H         //STC8A8K40S4A12
//#define  IAP_OFFSET      0xC000H         //STC8A8K48S4A12
//#define  IAP_OFFSET      0xE000H         //STC8A8K56S4A12
//#define  IAP_OFFSET      0xF000H         //STC8A8K60S4A12

void IapIdle()
{
    IAP_CONTR = 0;              //关闭 IAP 功能
    IAP_CMD = 0;                //清除命令寄存器
    IAP_TRIG = 0;               //清除触发寄存器
    IAP_ADDRH = 0x80;           //将地址设置到非 IAP 区域
    IAP_ADDRL = 0;
}

char IapRead(int addr)
{
    addr += IAP_OFFSET;         //使用 MOVC 读取 EEPROM 需要加上相应的偏移
    return *(char code *)(addr);//使用 MOVC 读取数据
}

void IapProgram(int addr, char dat)
{
    IAP_CONTR = WT_12M;         //使能 IAP
    IAP_CMD = 2;                //设置 IAP 写命令
    IAP_ADDRL = addr;           //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;      //设置 IAP 高地址
    IAP_DATA = dat;             //写 IAP 数据
    IAP_TRIG = 0x5a;            //写触发命令(0x5a)
    IAP_TRIG = 0xa5;            //写触发命令(0xa5)
    _nop_();
    IapIdle();                  //关闭 IAP 功能
}

void IapErase(int addr)
{
    IAP_CONTR = WT_12M;         //使能 IAP
    IAP_CMD = 3;                //设置 IAP 擦除命令
    IAP_ADDRL = addr;           //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;      //设置 IAP 高地址
    IAP_TRIG = 0x5a;            //写触发命令(0x5a)
    IAP_TRIG = 0xa5;            //写触发命令(0xa5)
    _nop_();                    //
    IapIdle();                  //关闭 IAP 功能
}

void main()
{
    IapErase(0x0400);
    P0 = IapRead(0x0400);       //P0=0xff
    IapProgram(0x0400, 0x12);
    P1 = IapRead(0x0400);       //P1=0x12

    while (1);
```

STC-MCU

```
}
```

15.5.3 使用串口送出EEPROM数据

汇编代码

```
AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H

IAP_DATA    DATA    0C2H
IAP_ADDRH   DATA    0C3H
IAP_ADDRL   DATA    0C4H
IAP_CMD     DATA    0C5H
IAP_TRIG    DATA    0C6H
IAP_CONTR   DATA    0C7H

WT_30M      EQU     80H
WT_24M      EQU     81H
WT_20M      EQU     82H
WT_12M      EQU     83H
WT_6M       EQU     84H
WT_3M       EQU     85H
WT_2M       EQU     86H
WT_1M       EQU     87H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H

UART_INIT:
            MOV     SCON,#5AH
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#15H
            RET

UART_SEND:
            JNB     TI,$
            CLR     TI
            MOV     SBUF,A
            RET

IAP_IDLE:
            MOV     IAP_CONTR,#0        ;关闭 IAP 功能
            MOV     IAP_CMD,#0          ;清除命令寄存器
            MOV     IAP_TRIG,#0         ;清除触发寄存器
            MOV     IAP_ADDRH,#80H      ;将地址设置到非 IAP 区域
            MOV     IAP_ADDRL,#0
            RET

IAP_READ:
            MOV     IAP_CONTR,#WT_12M   ;使能 IAP
            MOV     IAP_CMD,#1          ;设置 IAP 读命令
            MOV     IAP_ADDRL,DPL       ;设置 IAP 低地址
            MOV     IAP_ADDRH,DPH       ;设置 IAP 高地址
            MOV     IAP_TRIG,#5AH       ;写触发命令(0x5a)
            MOV     IAP_TRIG,#0A5H      ;写触发命令(0xa5)
```

```
                NOP
                MOV         A,IAP_DATA              ;读取 IAP 数据
                LCALL       IAP_IDLE                ;关闭 IAP 功能
                RET

IAP_PROGRAM:
                MOV         IAP_CONTR,#WT_12M       ;使能 IAP
                MOV         IAP_CMD,#2              ;设置 IAP 写命令
                MOV         IAP_ADDRL,DPL           ;设置 IAP 低地址
                MOV         IAP_ADDRH,DPH           ;设置 IAP 高地址
                MOV         IAP_DATA,A              ;写 IAP 数据
                MOV         IAP_TRIG,#5AH           ;写触发命令(0x5a)
                MOV         IAP_TRIG,#0A5H          ;写触发命令(0xa5)
                NOP
                LCALL       IAP_IDLE                ;关闭 IAP 功能
                RET

IAP_ERASE:
                MOV         IAP_CONTR,#WT_12M       ;使能 IAP
                MOV         IAP_CMD,#3              ;设置 IAP 擦除命令
                MOV         IAP_ADDRL,DPL           ;设置 IAP 低地址
                MOV         IAP_ADDRH,DPH           ;设置 IAP 高地址
                MOV         IAP_TRIG,#5AH           ;写触发命令(0x5a)
                MOV         IAP_TRIG,#0A5H          ;写触发命令(0xa5)
                NOP
                LCALL       IAP_IDLE                ;关闭 IAP 功能
                RET

MAIN:
                MOV         SP,#3FH

                LCALL       UART_INIT
                MOV         DPTR,#0400H
                LCALL       IAP_ERASE
                MOV         DPTR,#0400H
                LCALL       IAP_READ
                LCALL       UART_SEND
                MOV         DPTR,#0400H
                MOV         A,#12H
                LCALL       IAP_PROGRAM
                MOV         DPTR,#0400H
                LCALL       IAP_READ
                LCALL       UART_SEND

                SJMP        $

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

#define     FOSC            11059200UL
#define     BRT             (65536 - FOSC / 115200 / 4)

sfr         AUXR        =   0x8e;
sfr         T2H         =   0xd6;
sfr         T2L         =   0xd7;
```

```
sfr     IAP_DATA    =   0xC2;
sfr     IAP_ADDRH   =   0xC3;
sfr     IAP_ADDRL   =   0xC4;
sfr     IAP_CMD     =   0xC5;
sfr     IAP_TRIG    =   0xC6;
sfr     IAP_CONTR   =   0xC7;

#define WT_30M          0x80
#define WT_24M          0x81
#define WT_20M          0x82
#define WT_12M          0x83
#define WT_6M           0x84
#define WT_3M           0x85
#define WT_2M           0x86
#define WT_1M           0x87

void UartInit()
{
    SCON = 0x5a;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x15;
}

void UartSend(char dat)
{
    while (!TI);
    TI = 0;
    SBUF = dat;
}

void IapIdle()
{
    IAP_CONTR = 0;          //关闭 IAP 功能
    IAP_CMD = 0;            //清除命令寄存器
    IAP_TRIG = 0;           //清除触发寄存器
    IAP_ADDRH = 0x80;       //将地址设置到非 IAP 区域
    IAP_ADDRL = 0;
}

char IapRead(int addr)
{
    char dat;

    IAP_CONTR = WT_12M;     //使能 IAP
    IAP_CMD = 1;            //设置 IAP 读命令
    IAP_ADDRL = addr;       //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;  //设置 IAP 高地址
    IAP_TRIG = 0x5a;        //写触发命令(0x5a)
    IAP_TRIG = 0xa5;        //写触发命令(0xa5)
    _nop_();
    dat = IAP_DATA;         //读 IAP 数据
    IapIdle();              //关闭 IAP 功能

    return dat;
}

void IapProgram(int addr, char dat)
```

```
{
	IAP_CONTR = WT_12M;		//使能 IAP
	IAP_CMD = 2;			//设置 IAP 写命令
	IAP_ADDRL = addr;		//设置 IAP 低地址
	IAP_ADDRH = addr >> 8;		//设置 IAP 高地址
	IAP_DATA = dat;			//写 IAP 数据
	IAP_TRIG = 0x5a;		//写触发命令(0x5a)
	IAP_TRIG = 0xa5;		//写触发命令(0xa5)
	_nop_();
	IapIdle();			//关闭 IAP 功能
}

void IapErase(int addr)
{
	IAP_CONTR = WT_12M;		//使能 IAP
	IAP_CMD = 3;			//设置 IAP 擦除命令
	IAP_ADDRL = addr;		//设置 IAP 低地址
	IAP_ADDRH = addr >> 8;		//设置 IAP 高地址
	IAP_TRIG = 0x5a;		//写触发命令(0x5a)
	IAP_TRIG = 0xa5;		//写触发命令(0xa5)
	_nop_();			//
	IapIdle();			//关闭 IAP 功能
}

void main()
{
	UartInit();
	IapErase(0x0400);
	UartSend(IapRead(0x0400));
	IapProgram(0x0400, 0x12);
	UartSend(IapRead(0x0400));

	while (1);
}
```

STC MCU

16 ADC模数转换，内部 1.344V参考信号源

STC8A 系列单片机内部集成了一个 12 位高速 A/D 转换器。ADC 的时钟频率为系统频率 2 分频再经过用户设置的分频系数进行再次分频（ADC 的工作时钟频率范围为 SYSclk/2/1 到 SYSclk/2/16）。

STC8G 系列的 ADC 最快速度：**12 位 ADC 为 800K（每秒进行 80 万次 ADC 转换），10 位 ADC 为 500K（每秒进行 50 万次 ADC 转换）**

ADC 转换结果的数据格式有两种：左对齐和右对齐。可方便用户程序进行读取和引用。

注意：ADC 的第 15 通道只能用于检测内部 1.344V 参考信号源，参考电压值出厂时校准为 1.344V，由于制造误差以及测量误差，导致实际的内部参考电压相比 1.344V，大约有±1%的误差。如果用户需要知道每一颗芯片的准确内部参考电压值，可外接精准参考电压，然后利用 ADC 的第 15 通道进行测量标定。

芯片与 ADC 相关的管脚 ADC_AVcc/ADC_Gnd/ADC_VRef+，一定不能浮空，ADC_AVcc 需要连接到 Vcc，ADC_AGnd 需要连接到 Gnd，ADC_VRef+必须接外部参考电源或者直接连到 Vcc

特别注意：

对于 STC8A 系列的芯片，当用户使用 ADC_CONTR.7（ADC_POWER）寄存器打开 ADC 模块的电源时，ADC 模块会强制将 ADC_CONTR 寄存器中 ADC_CHS 所对于的 I/O 口设置为高阻输入模式，并关闭此端口上的数字输入通道。

例如：用户软件中执行完成“ADC_CONTR = 0x80”语句后，ADC 模块的电源被打开的同时，ADC 的通道 0 所对于的端口 P1.0 也会被强制设置为高阻输入模式，并关闭 P1.0 上所有的数字功能，包括 P1.0 口数字电平的输入输出、PWM0 的输出以及串口 2 的输入，所以此时 P1.0 口上将无法接收串口 2 的数据，也无法输出 PWM 波形。

STC8A 系列单片机的 15 路 ADC 所对应的 I/O 口均有多个复用的数字功能，所以为了避免打开 ADC 电源的同时影响不相关的 I/O 的数字功能，**软件中在设置 ADC_CONTR 寄存器时，务必将 ADC_POWER 和 ADC_CHS 同时设置**。比如用户系统中需要使用 P1.5 口进行 ADC 转换，则对 ADC_CONTR 进行初始化时请直接使用“ADC_CONTR = 0x85”语句进行初始化。

16.1 ADC相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]				000x,0000
ADC_RES	ADC 转换结果高位寄存器	BDH									0000,0000
ADC_RESL	ADC 转换结果低位寄存器	BEH									0000,0000
ADCCFG	ADC 配置寄存器	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
ADCTIM	ADC 时序控制寄存器	FEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]					0010,1010

16.1.1 ADC控制寄存器（ADC_CONTR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]			

ADC_POWER：ADC 电源控制位

0：关闭 ADC 电源

1：打开 ADC 电源。

建议进入空闲模式和掉电模式前将 ADC 电源关闭，以降低功耗

ADC_START：ADC 转换启动控制位。写入 1 后开始 ADC 转换，转换完成后硬件自动将此位清零。

0：无影响。即使 ADC 已经开始转换工作，写 0 也不会停止 A/D 转换。

1：开始 ADC 转换，转换完成后硬件自动将此位清零。

ADC_FLAG：ADC 转换结束标志位。当 ADC 完成一次转换后，硬件会自动将此位置 1，并向 CPU 提出中断请求。此标志位必须软件清零。

STC MCU

ADC_CHS[3:0]：ADC 模拟通道选择位

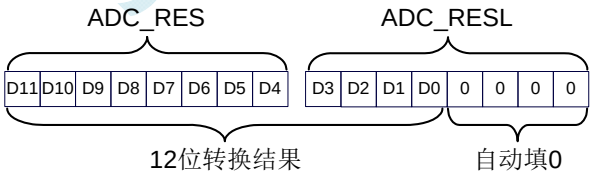
ADC_CHS[3:0]	ADC 通道
0000	P1.0/ADC0
0001	P1.1/ADC1
0010	P1.2/ADC2
0011	P1.3/ADC3
0100	P1.4/ADC4
0101	P1.5/ADC5
0110	P1.6/ADC6
0111	P1.7/ADC7
1000	P0.0/ADC8
1001	P0.1/ADC9
1010	P0.2/ADC10
1011	P0.3/ADC11
1100	P0.4/ADC12
1101	P0.5/ADC13
1110	P0.6/ADC14
1111	测试内部 1.344V 的 REFV 电压

16.1.2 ADC配置寄存器（ADCCFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCCFG	DEH	-	-	RESFMT	-	SPEED[3:0]			

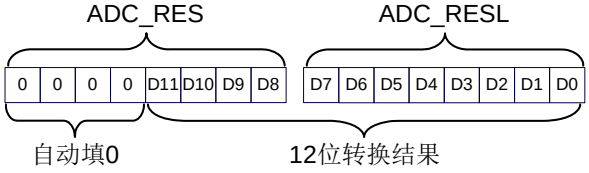
RESFMT：ADC 转换结果格式控制位

0：转换结果左对齐。ADC_RES 保存结果的高 8 位，ADC_RESL 保存结果的低 2 位。格式如下：



RESFMT=0

1：转换结果右对齐。ADC_RES 保存结果的高 2 位，ADC_RESL 保存结果的低 8 位。格式如下：



RESFMT=1

SPEED[3:0]：ADC 时钟控制（F_{ADC}＝SYSclk/2/16/（SPEED[3:0]+1））

SPEED[3:0]	ADC 转换时间（CPU 时钟数）	SPEED[3:0]	ADC 转换时间（CPU 时钟数）
0000	32	1000	288
0001	64	1001	320
0010	96	1010	352
0011	128	1011	384
0100	160	1100	416
0101	192	1101	448
0110	224	1110	480
0111	256	1111	512

例如：系统时钟为 12MHz，SPEED[3:0]设置为 3（0011B），则 ADC 转换速度为
F_{ADC}＝SYSclk/2/16/（SPEED[3:0]+1）＝12MHz/2/16/（3+1）＝93.75KHz

16.1.3 ADC转换结果寄存器（ADC_RES，ADC_RESL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDH								
ADC_RESL	BEH								

当 A/D 转换完成后，10 位的转换结果会自动保存到 ADC_RES 和 ADC_RESL 中。保存结果的数据格式请参考 ADC_CFG 寄存器中的 RESFMT 设置。

STC MCU

16.2 ADC相关计算公式

16.2.1 ADC转换结果计算公式

$$12\text{位ADC转换结果} = 4096 \times \frac{\text{ADC被转换通道的输入电压Vin}}{\text{ADC外部参考源的电压}}$$

16.2.2 反推ADC输入电压计算公式

$$\text{ADC被转换通道的输入电压Vin} = \text{ADC外部参考源的电压} \times \frac{\text{12位ADC转换结果}}{4096}$$

16.2.3 反推工作电压计算公式

当需要使用 ADC 输入电压和 ADC 转换结果反推工作电压时，目标芯片的 ADC_Vref+管脚必须连接到 Vcc 管脚。

$$\text{MCU工作电压Vcc} = 4096 \times \frac{\text{ADC被转换通道的输入电压Vin}}{\text{12位ADC转换结果}}$$

16.3 12 位ADC静态特性

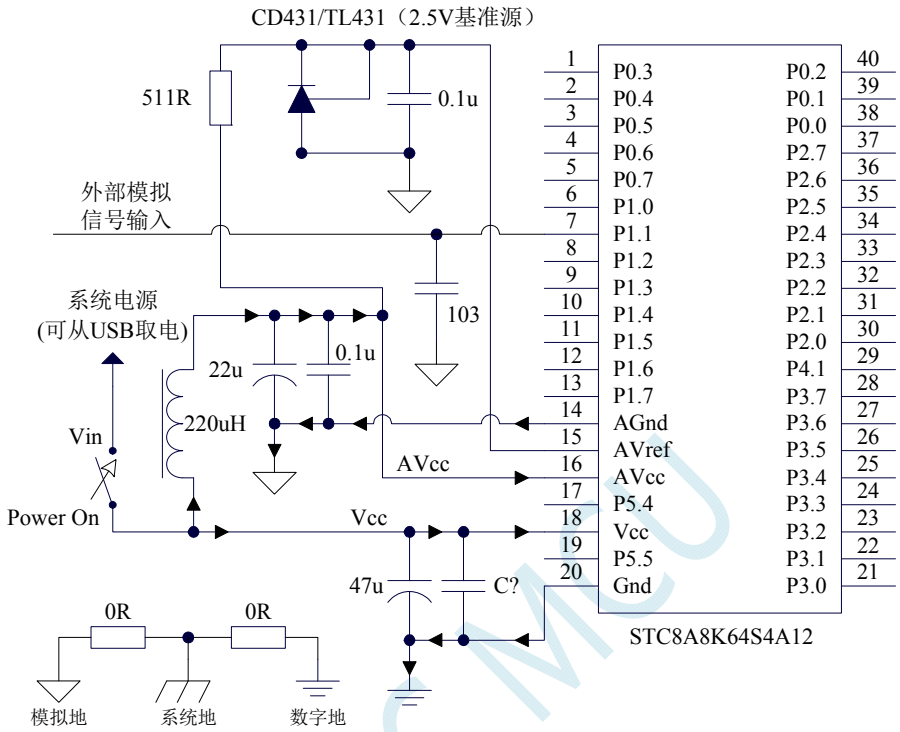
符号	描述	最小值	典型值	最大值	单位
RES	分辨率	-	12	-	Bits
E_T	整体误差	-	0.5	1	LSB
E_O	偏移误差	-	-0.1	1	LSB
E_G	增益误差	-	0	1	LSB
E_D	微分非线性误差	-	0.7	1.5	LSB
E_I	积分非线性误差	-	1	2	LSB
R_{AIN}	通道等效电阻	-	∞	-	ohm
R_{ESD}	采样保持电容前串接的抗静电电阻	-	700	-	ohm
C_{ADC}	内部采样保持电容	-	16.5	-	pF

注意事项：

1. **使用 ADC 时，要注意下面的要求：AVCC 跟 VCC 电压差不要超过 0.1V，VREF 在 2.0~AVCC 之间，P1.0~P1.7 P0.0~P0.6 电压不能比 AVCC 高。**
2. **使用外部晶振时，要注意下面的要求： AVCC 跟 VCC 电压差不要超过 0.1V，P1.0~P1.7 P0.0~P0.6 电压不能比 AVCC 高，否则晶振不起振，形同死机。**

16.4 ADC应用参考线路图

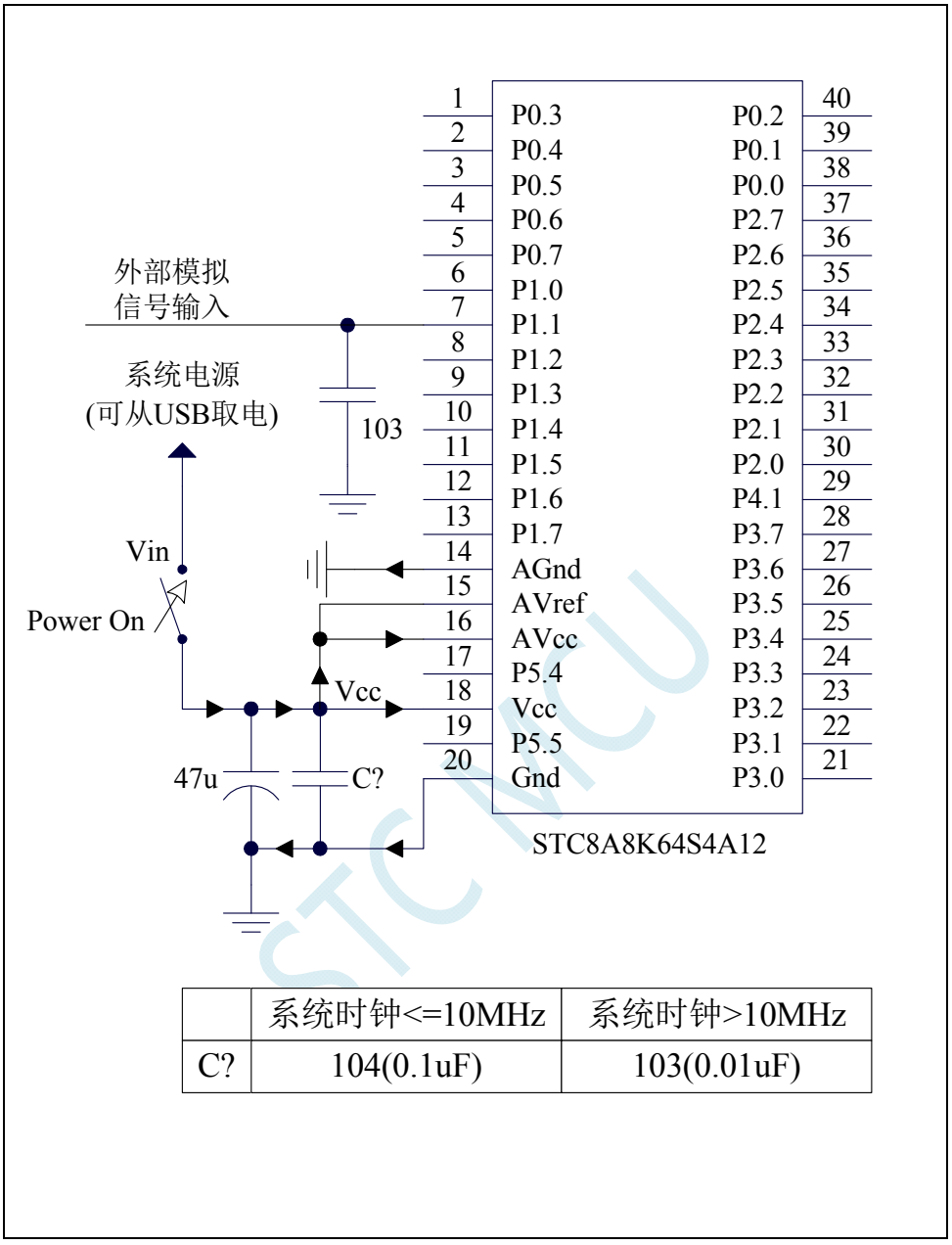
16.4.1 高精度ADC应用



	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

STC MCU

16.4.2 ADC一般应用（对ADC精度要求不高的应用）



	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

16.5 范例程序

16.5.1 ADC基本操作（查询方式）

汇编代码

```
;测试工作频率为11.0592MHz

ADC_CONTR   DATA    0BCH
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH

P1M0        DATA    092H
P1M1        DATA    091H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH

            MOV     P1M0,#00H           ;设置P1.0为ADC口
            MOV     P1M1,#01H
            MOV     ADCCFG,#0FH         ;设置ADC时钟为系统时钟/2/16/16
            MOV     ADC_CONTR,#80H      ;使能ADC模块
LOOP:
            ORL     ADC_CONTR,#40H      ;启动AD转换
            NOP
            NOP
            MOV     A,ADC_CONTR         ;查询ADC完成标志
            JNB     ACC.5,$-2
            ANL     ADC_CONTR,#NOT 20H  ;清完成标志
            MOV     P2,ADC_RES          ;读取ADC结果

            SJMP    LOOP

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为11.0592MHz

sfr     ADC_CONTR   =   0xbc;
sfr     ADC_RES     =   0xbd;
sfr     ADC_RESL    =   0xbe;
sfr     ADCCFG      =   0xde;

sfr     P1M0        =   0x92;
sfr     P1M1        =   0x91;

void main()
{
```

```
    P1M0 = 0x00;                    //设置 P1.0 为 ADC 口
    P1M1 = 0x01;
    ADCCFG = 0x0f;                  //设置 ADC 时钟为系统时钟/2/16/16
    ADC_CONTR = 0x80;               //使能 ADC 模块

    while (1)
    {
        ADC_CONTR |= 0x40;          //启动 AD 转换
        _nop_();
        _nop_();
        while (!(ADC_CONTR & 0x20)); //查询 ADC 完成标志
        ADC_CONTR &= ~0x20;         //清完成标志
        P2 = ADC_RES;               //读取 ADC 结果
    }
}
```

16.5.2 ADC基本操作（中断方式）

汇编代码

```
;测试工作频率为 11.0592MHz

ADC_CONTR   DATA    0BCH
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH

EADC        BIT     IE.5

P1M0        DATA    092H
P1M1        DATA    091H

            ORG     0000H
            LJMP    MAIN
            ORG     002BH
            LJMP    ADCISR

            ORG     0100H
ADCISR:
            ANL     ADC_CONTR,#NOT 20H      ;清完成标志
            MOV     P2,ADC_RES              ;读取 ADC 结果
            ORL     ADC_CONTR,#40H          ;继续 AD 转换
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     P1M0,#00H               ;设置 P1.0 为 ADC 口
            MOV     P1M1,#01H
            MOV     ADCCFG,#0FH             ;设置 ADC 时钟为系统时钟/2/16/16
            MOV     ADC_CONTR,#80H          ;使能 ADC 模块
            SETB    EADC                    ;使能 ADC 中断
            SETB    EA
            ORL     ADC_CONTR,#40H          ;启动 AD 转换

            SJMP    $

            END
```

STC MCU

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为 11.0592MHz

sfr         ADC_CONTR   =   0xbc;
sfr         ADC_RES     =   0xbd;
sfr         ADC_RESL    =   0xbe;
sfr         ADCCFG      =   0xde;

sbit        EADC        =   IE^5;

sfr         P1M0        =   0x92;
sfr         P1M1        =   0x91;

void ADC_Isr() interrupt 5
{
    ADC_CONTR &= ~0x20;             //清中断标志
    P2 = ADC_RES;                   //读取 ADC 结果
    ADC_CONTR |= 0x40;              //继续 AD 转换
}

void main()
{
    P1M0 = 0x00;                    //设置 P1.0 为 ADC 口
    P1M1 = 0x01;
    ADCCFG = 0x0f;                  //设置 ADC 时钟为系统时钟/2/16/16
    ADC_CONTR = 0x80;               //使能 ADC 模块
    EADC = 1;                       //使能 ADC 中断
    EA = 1;
    ADC_CONTR |= 0x40;              //启动 AD 转换

    while (1);
}
```

16.5.3 格式化ADC转换结果

汇编代码

```
;测试工作频率为 11.0592MHz

ADC_CONTR   DATA    0BCH
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH

P1M0        DATA    092H
P1M1        DATA    091H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH
```

```
                MOV     P1M0,#00H               ;设置 P1.0 为 ADC 口
                MOV     P1M1,#01H
                MOV     ADCCFG,#0FH             ;设置 ADC 时钟为系统时钟/2/16/16
                MOV     ADC_CONTR,#80H          ;使能 ADC 模块

                ORL     ADC_CONTR,#40H          ;启动 AD 转换
                NOP
                NOP
                MOV     A,ADC_CONTR             ;查询 ADC 完成标志
                JNB     ACC.5,$-2
                ANL     ADC_CONTR,#NOT 20H      ;清完成标志

                MOV     ADCCFG,#00H             ;设置结果左对齐
                MOV     A,ADC_RES               ;A 存储 ADC 的 12 位结果的高 8 位
                MOV     B,ADC_RESL              ;B[7:4]存储 ADC 的 12 位结果的低 4 位,B[3:0]为 0

;               MOV     ADCCFG,#20H             ;设置结果右对齐
;               MOV     A,ADC_RES               ;A[3:0]存储 ADC 的 12 位结果的高 4 位,A[7:4]为 0
;               MOV     B,ADC_RESL              ;B 存储 ADC 的 12 位结果的低 8 位

                SJMP    $

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为 11.0592MHz

sfr     ADC_CONTR   =   0xbc;
sfr     ADC_RES     =   0xbd;
sfr     ADC_RESL    =   0xbe;
sfr     ADCCFG      =   0xde;

sfr     P1M0        =   0x92;
sfr     P1M1        =   0x91;

void main()
{
    P1M0 = 0x00;                    //设置 P1.0 为 ADC 口
    P1M1 = 0x01;
    ADCCFG = 0x0f;                  //设置 ADC 时钟为系统时钟/2/16/16
    ADC_CONTR = 0x80;               //使能 ADC 模块
    ADC_CONTR |= 0x40;              //启动 AD 转换
    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20));    //查询 ADC 完成标志
    ADC_CONTR &= ~0x20;             //清完成标志

    ADCCFG = 0x00;                  //设置结果左对齐
    ACC = ADC_RES;                  //A 存储 ADC 的 12 位结果的高 8 位
    B = ADC_RESL;                   //B[7:4]存储 ADC 的 12 位结果的低 4 位,B[3:0]为 0

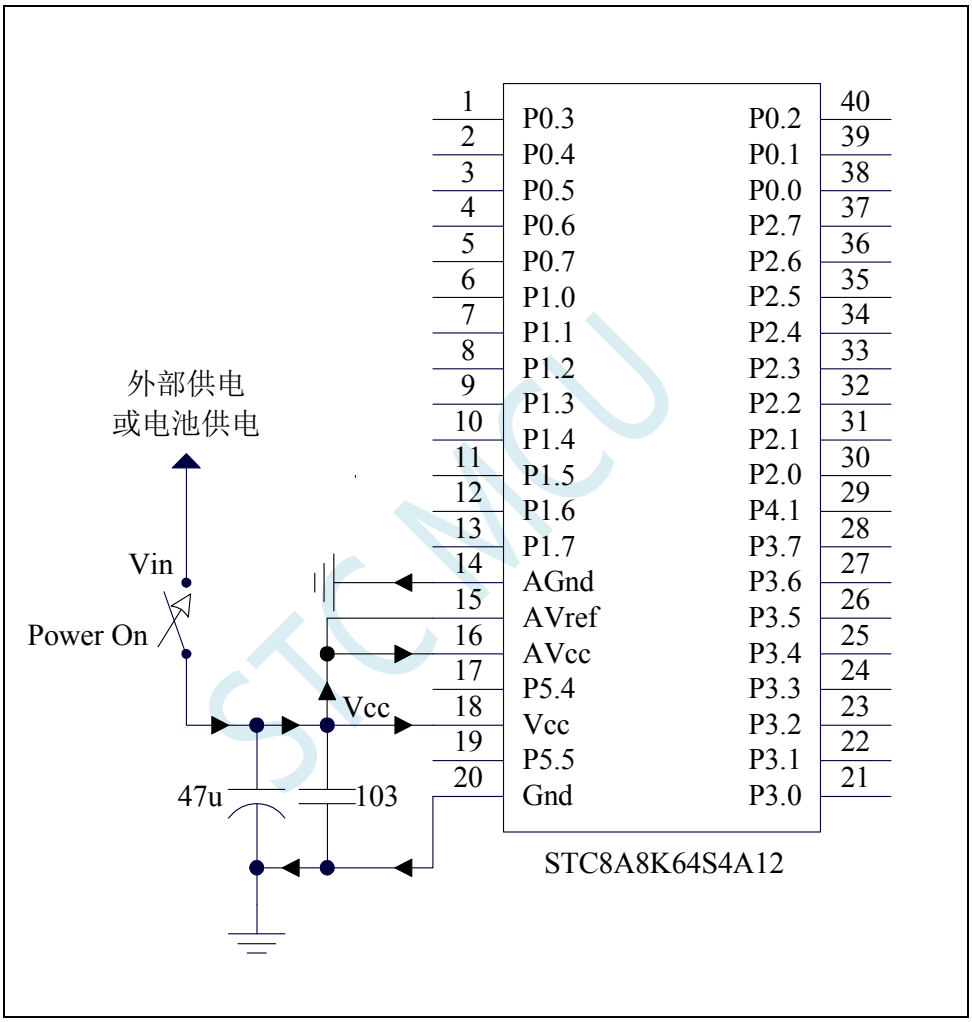
//  ADCCFG = 0x20;                  //设置结果右对齐
//  ACC = ADC_RES;                  //A[3:0]存储 ADC 的 12 位结果的高 4 位,A[7:4]为 0
//  B = ADC_RESL;                   //B 存储 ADC 的 12 位结果的低 8 位
```

```
    while (1);
}
```

16.5.4 利用ADC第 15 通道测量外部电压或电池电压

STC8A 系列 ADC 的第 15 通道用于测量内部参考信号源，由于内部参考信号源很稳定，约为 1.344V，且不会随芯片的工作电压的改变而变化，所以可以通过测量内部 1.344V 参考信号源，然后通过 ADC 的值便可反推出外部电压或外部电池电压。

下图为参考线路图:



C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为 11.0592MHz

#define    FOSC        11059200UL
#define    BRT         (65536 - FOSC / 115200 / 4)

sfr        AUXR        =    0x8e;
```

```
sfr         ADC_CONTR   =   0xbc;
sfr         ADC_RES     =   0xbd;
sfr         ADC_RESL    =   0xbe;
sfr         ADCCFG      =   0xde;

sfr         P1M0        =   0x92;
sfr         P1M1        =   0x91;

int         *BGV;                   //内部 Bandgap 电压值存放在 idata 中
                                    //idata 的 EFH 地址存放高字节
                                    //idata 的 F0H 地址存放低字节
                                    //电压单位为毫伏(mV)
bit         busy;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void ADCInit()
{
    ADCCFG = 0x2f;              //设置 ADC 时钟为系统时钟/2/16/16
    ADC_CONTR = 0x8f;           //使能 ADC 模块,并选择第 16 通道
}

int   ADCRead()
{
    int res;

    ADC_CONTR |= 0x40;          //启动 AD 转换
    _nop_();
    _nop_();
```

STC MCU

```
    while (!(ADC_CONTR & 0x20));        //查询 ADC 完成标志
    ADC_CONTR &= ~0x20;                 //清完成标志
    res = (ADC_RES << 8) | ADC_RESL;    //读取 ADC 结果

    return res;
}

void main()
{
    int res;
    int vcc;
    int i;

    BGV = (int idata *)0xef;
    ADCInit();                          //ADC 初始化
    UartInit();                         //串口初始化

    ES = 1;
    EA = 1;

    ADCRead();
    ADCRead();                          //前两个数据丢弃
    res = 0;
    for (i=0; i<8; i++)
    {
        res += ADCRead();               //读取 8 次数据
    }
    res >>= 3;                          //取平均值

    vcc = (int)(4095L * *BGV / res);    //计算 VREF 管脚电压,即电池电压
                                        //注意,此电压的单位为毫伏(mV)
    UartSend(vcc >> 8);                 //输出电压值到串口
    UartSend(vcc);

    while (1);
}
```

16.5.5 ADC做电容感应触摸按键

按键是电路最常用的零件之一，是人机界面重要的输入方式，我们最熟悉的是机械式按键，但是机械按键有一个缺点（特别是便宜的按键），触点有寿命，很容易出现接触不良而失效。而非接触的按键则没有机械触点，寿命长，使用方便。

非接触的按键有多种方案，而电容感应按键则是低成本的方案，多年前一般是使用专门的 IC 来实现，随着 MCU 功能的加强，以及广大用户的实践经验，直接使用 MCU 来做电容感应按键的技术已经成熟，其中最典型最可靠的是使用 ADC 做的方案。

本文档详述使用 STC 带 ADC 的系列 MCU 做的方案，可以使用任何带 ADC 功能的 MCU 来实现。下面前 3 个图是用得最多的方式，原理都一样，本文使用第 2 个图。

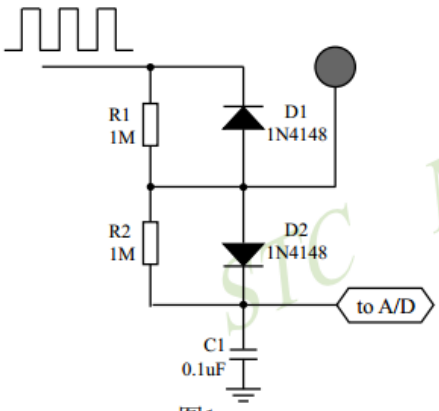


图1

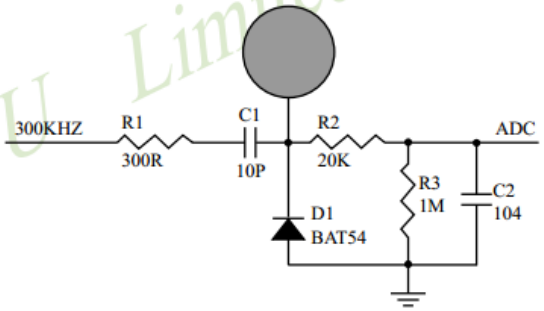


图2

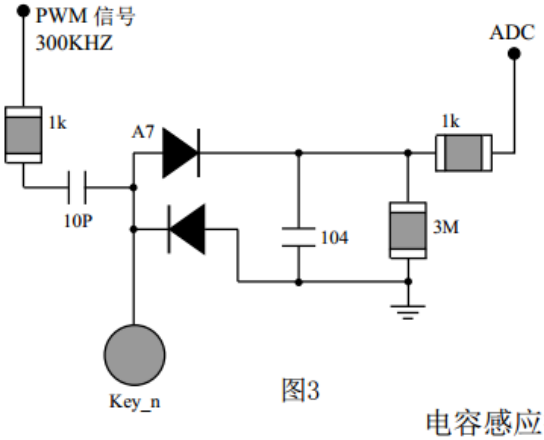


图3

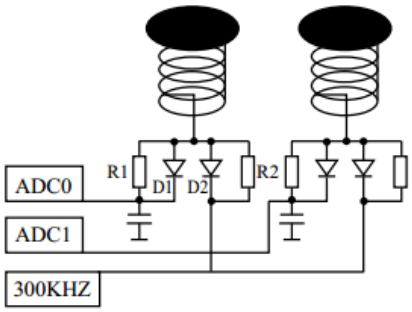
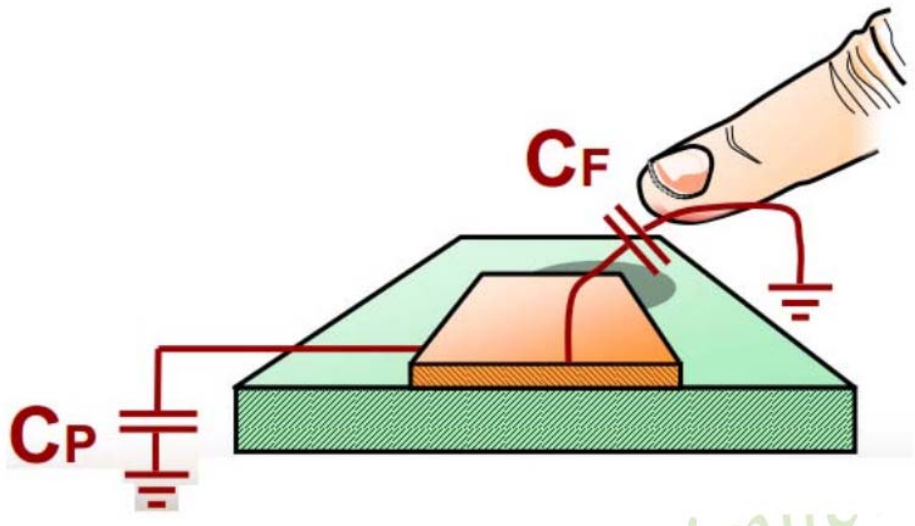


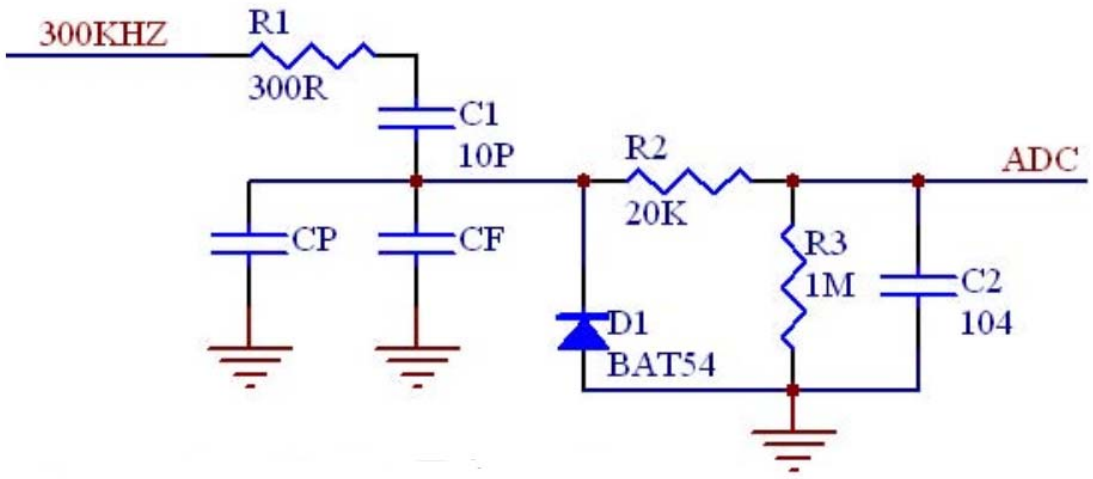
图4 加了感应弹簧

电容感应按键取样电路

一般实际应用时，都使用图 4 所示的感应弹簧来加大手指按下的面积。感应弹簧等效一块对地的金属板，对地有一个电容 CP，而手指按下后，则再并联一个对地的电容 CF，如下图所示。



下面为电路图的说明，CP 为金属板和分布电容，CF 为手指电容，并联在一起与 C1 对输入的 300KHZ 方波进行分压，经过 D1 整流，R2、C2 滤波后送 ADC，当手指压上去后，送去 ADC 的电压降低，程序就可以检测出按键动作。



C 语言代码

```
//测试工作频率为24MHz

#include "reg51.h"
#include "intrins.h"

#define    MAIN_Fosc     24000000UL                  //定义主时钟
#define    Timer0_Reload (65536UL -(MAIN_Fosc / 600000))  //Timer 0 重装值，对应300KHZ

typedef    unsigned char  u8;
typedef    unsigned int   u16;
typedef    unsigned long  u32;

sfr        P0M1        =    0x93;
sfr        P0M0        =    0x94;
sfr        P1M1        =    0x91;
sfr        P1M0        =    0x92;
sfr        P2M1        =    0x95;
sfr        P2M0        =    0x96;
sfr        P3M1        =    0xb1;
sfr        P3M0        =    0xb2;
sfr        P4M1        =    0xb3;
sfr        P4M0        =    0xb4;
sfr        P5M1        =    0xc9;
sfr        P5M0        =    0xca;

sfr        ADC_CONTR   =    0xBC;                  //带AD系列
sfr        ADC_RES     =    0xBD;                  //带AD系列
sfr        ADC_RESL    =    0xBE;                  //带AD系列
sfr        AUXR        =    0x8E;
sfr        AUXR2       =    0x8F;

#define    CHANNEL     8                           //ADC 通道数
#define    ADC_90T     (3<<5)                      //ADC 时间 90T
#define    ADC_180T    (2<<5)                      //ADC 时间 180T
#define    ADC_360T    (1<<5)                      //ADC 时间 360T
#define    ADC_540T    0                           //ADC 时间 540T
#define    ADC_FLAG    (1<<4)                      //软件清0
#define    ADC_START   (1<<3)                      //自动清0

sbit       P_LED7      =    P2^7;
```

```
sbit    P_LED6    =    P2^6;
sbit    P_LED5    =    P2^5;
sbit    P_LED4    =    P2^4;
sbit    P_LED3    =    P2^3;
sbit    P_LED2    =    P2^2;
sbit    P_LED1    =    P2^1;
sbit    P_LED0    =    P2^0;

u16 idata adc[TOUCH_CHANNEL];                   //当前 ADC 值
u16 idata adc_prev[TOUCH_CHANNEL];              //上一个 ADC 值
u16 idata TouchZero[TOUCH_CHANNEL];             //0 点 ADC 值
u8 idata TouchZeroCnt[TOUCH_CHANNEL];           //0 点自动跟踪计数
u8 cnt_250ms;

void delay_ms(u8 ms);
void ADC_init(void);
u16 Get_ADC10bitResult(u8 channel);
void AutoZero(void);
u8 check_adc(u8 index);
void ShowLED(void);

void main(void)
{
    u8 i;

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    delay_ms(50);
    ET0 = 0;                                    //初始化 Timer0 输出一个 300KHZ 时钟
    TR0 = 0;
    AUXR |= 0x80;                               //Timer0 set as 1T mode
    AUXR2 |= 0x01;                              //允许输出时钟
    TMOD = 0;                                   //Timer0 set as Timer, 16 bits Auto Reload.
    TH0 = (u8)(Timer0_Reload >> 8);
    TL0 = (u8)Timer0_Reload;
    TR0 = 1;
    ADC_init();                                 //ADC 初始化
    delay_ms(50);                               //延时 50ms
    for (i=0; i<TOUCH_CHANNEL; i++)             //初始化 0 点和上一个值和 0 点自动跟踪计数
    {
        adc_prev[i] = 1023;
        TouchZero[i] = 1023;
        TouchZeroCnt[i] = 0;
    }
    cnt_250ms = 0;
    while (1)
    {
        delay_ms(50);                           //每隔 50ms 处理一次按键
```

STC MCU

```
            ShowLED();
            if (++cnt_250ms >= 5)
            {
                cnt_250ms = 0;
                AutoZero();                             //每隔 250ms 处理一次 0 点自动跟踪
            }
        }
}

void delay_ms(u8 ms)
{
        unsigned int i;

        do
        {
            i = MAIN_Fosc / 13000;
            while(--i) ;
        } while(--ms);
}

void ADC_init(void)
{
        P1M0 = 0x00;                                    //8 路 ADC
        P1M1 = 0xff;
        ADC_CONTR = 0x80;                               //允许 ADC
}

u16 Get_ADC10bitResult(u8 channel)
{
        ADC_RES = 0;
        ADC_RESL = 0;
        ADC_CONTR = 0x80 | ADC_90T | ADC_START | channel;   //触发 ADC
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        while((ADC_CONTR & ADC_FLAG) == 0) ;            //等待 ADC 转换结束
        ADC_CONTR = 0x80;                               //清除标志
        return(((u16)ADC_RES << 2) | ((u16)ADC_RESL & 3));  //返回 ADC 结果
}

void AutoZero(void)                                     //250ms 调用一次
                                                        //这是使用相邻 2 个采样的差的绝对值之和来检测。
{
        u8 i;
        u16 j,k;

        for(i=0; i<TOUCH_CHANNEL; i++)                  //处理 8 个通道
        {
            j = adc[i];
            k = j - adc_prev[i];                        //减前一个读数
            F0 = 0;                                     //按下
            if(k & 0x8000) F0 = 1, k = 0 - k;           //释放 求出两次采样的差值
            if(k >= 20)                                 //变化比较大
            {
                TouchZeroCnt[i] = 0;                    //如果变化比较大，则清 0 计数器
                if(F0) TouchZero[i] = j;                //如果是释放，并且变化比较大，则直接替代
            }
            else                                        //变化比较小，则蠕动，自动 0 点跟踪
```

```
            {
                if(++TouchZeroCnt[i] >= 20)                 //连续检测到小变化 20 次/4 = 5 秒.
                {
                    TouchZeroCnt[i] = 0;
                    TouchZero[i] = adc_prev[i];             //变化缓慢的值作为 0 点                 }
            }
            adc_prev[i] = j;                                //保存这一次的采样值
        }
}

u8 check_adc(u8 index)                                      //获取触摸信息函数 50ms 调用 1 次
                                                            //判断键按下或释放,有回差控制
{
    u16 delta;

    adc[index] = 1023 - Get_ADC10bitResult(index);          //获取 ADC 值, 转成按下键, ADC 值增加
    if(adc[index] < TouchZero[index]) return 0;             //比 0 点还小的值，则认为是键释放
    delta = adc[index] - TouchZero[index];
    if(delta >= 40) return 1;                               //键按下
    if(delta <= 20) return 0;                               //键释放
    return 2;                                               //保持原状态
}

void ShowLED(void)
{
    u8 i;

    i = check_adc(0);
    if(i == 0) P_LED0 = 1;                                  //指示灯灭
    if(i == 1) P_LED0 = 0;                                  //指示灯亮
    i = check_adc(1);
    if(i == 0) P_LED1 = 1;                                  //指示灯灭
    if(i == 1) P_LED1 = 0;                                  //指示灯亮
    i = check_adc(2);
    if(i == 0) P_LED2 = 1;                                  //指示灯灭
    if(i == 1) P_LED2 = 0;                                  //指示灯亮
    i = check_adc(3);
    if(i == 0) P_LED3 = 1;                                  //指示灯灭
    if(i == 1) P_LED3 = 0;                                  //指示灯亮
    i = check_adc(4);
    if(i == 0) P_LED4 = 1;                                  //指示灯灭
    if(i == 1) P_LED4 = 0;                                  //指示灯亮
    i = check_adc(5);
    if(i == 0) P_LED5 = 1;                                  //指示灯灭
    if(i == 1) P_LED5 = 0;                                  //指示灯亮
    i = check_adc(6);
    if(i == 0) P_LED6 = 1;                                  //指示灯灭
    if(i == 1) P_LED6 = 0;                                  //指示灯亮
    i = check_adc(7);
    if(i == 0) P_LED7 = 1;                                  //指示灯灭
    if(i == 1) P_LED7 = 0;                                  //指示灯亮
}
```

汇编代码

```
;测试工作频率为 24MHz

Fosc_KHZ    EQU     24000                                   ;定义主时钟 KHZ
```

```
Reload      EQU     (65536 - Fosc_KHZ/600)      ;Timer 0  重装值,对应 300KHZ

ADC_CONTR   DATA    0xBC                        ;带 AD 系列
ADC_RES     DATA    0xBD                        ;带 AD 系列
ADC_RESL    DATA    0xBE                        ;带 AD 系列
AUXR        DATA    0x8E
AUXR2       DATA    0x8F

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

CHANNEL     EQU     8                           ;ADC 通道数
ADC_90T     EQU     (3 SHL 5)                   ;ADC 时间 90T
ADC_180T    EQU     (2 SHL 5)                   ;ADC 时间 180T
ADC_360T    EQU     (1 SHL 5)                   ;ADC 时间 360T
ADC_540T    EQU     0                           ;ADC 时间 540T
ADC_FLAG    EQU     (1 SHL 4)                   ;软件清 0
ADC_START   EQU     (1 SHL 3)                   ;自动清 0

P_LED7      BIT     P2.7;
P_LED6      BIT     P2.6;
P_LED5      BIT     P2.5;
P_LED4      BIT     P2.4;
P_LED3      BIT     P2.3;
P_LED2      BIT     P2.2;
P_LED1      BIT     P2.1;
P_LED0      BIT     P2.0;
adc         EQU     30H                         ;当前 ADC 值  30H~3FH,两字节一个值
adc_prev    EQU     40H                         ;上一个 ADC 值  40H~4FH,两字节一个值
TouchZero   EQU     50H                         ;0 点 ADC 值  50H~5FH,两字节一个值
TouchZeroCnt EQU    60H                         ;0 点自动跟踪计数  60H~67H
cnt_250ms   DATA    68H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#0D0H
            MOV     P0M0,#00H
            MOV     P0M1,#00H
            MOV     P1M0,#00H
            MOV     P1M1,#00H
            MOV     P2M0,#00H
            MOV     P2M1,#00H
            MOV     P3M0,#00H
            MOV     P3M1,#00H
            MOV     P4M0,#00H
            MOV     P4M1,#00H
```

```
                MOV     P5M0,#00H
                MOV     P5M1,#00H

                MOV     R7,#50
                LCALL   F_delay_ms
                CLR     ET0                     ;初始化 Timer0 输出一个 300KHZ 时钟
                CLR     TR0
                ORL     AUXR,#080H              ;Timer0 set as 1T mode
                ORL     AUXR2,#01H              ;允许输出时钟
                MOV     TMOD,#0                 ;Timer0 set as Timer,16 bits Auto Reload.
                MOV     TH0,#HIGH Reload
                MOV     TL0,#LOW Reload
                SETB    TR0
                LCALL   F_ADC_init
                MOV     R7,#50
                LCALL   F_delay_ms
                MOV     R0,#adc_prev            ;初始化上一个 ADC 值
L_Init_Loop1:
                MOV     @R0,#03H
                INC     R0
                MOV     @R0,#0FFH
                INC     R0
                MOV     A,R0
                CJNE    A,#(adc_prev + CHANNEL * 2),L_Init_Loop1
                MOV     R0,#TouchZero           ;初始化 0 点 ADC 值
L_Init_Loop2:
                MOV     @R0,#03H
                INC     R0
                MOV     @R0,#0FFH
                INC     R0
                MOV     A,R0
                CJNE    A,#(TouchZero+CHANNEL * 2),L_Init_Loop2
                MOV     R0,#TouchZeroCnt        ;初始化自动跟踪计数值
L_Init_Loop3:
                MOV     @R0,#0
                INC     R0
                MOV     A,R0
                CJNE    A,#(TouchZeroCnt + CHANNEL),L_Init_Loop3
                MOV     cnt_250ms,#5
L_MainLoop:
                MOV     R7,#50                  ;延时 50ms
                LCALL   F_delay_ms
                LCALL   F_ShowLED               ;处理一次触摸键值
                DJNZ    cnt_250ms,L_MainLoop
                MOV     cnt_250ms,#5            ;250ms 处理一次 0 点自动跟踪
                LCALL   F_AutoZero              ;自动跟踪零点
                SJMP    L_MainLoop

F_ADC_init:
                MOV     P1M0,#00H               ;8 路 ADC
                MOV     P1M1,#0FFH
                MOV     ADC_CONTR,#080H         ;允许 ADC
                RET

F_Get_ADC10bitResult:
                MOV     ADC_RES,#0
                MOV     ADC_RESL,#0
                MOV     A,R7
                ORL     A,#0E8H                 ;触发 ADC
```

```
                MOV     ADC_CONTR,A
                NOP
                NOP
                NOP
                NOP
L_10bitADC_Loop1:
                MOV     A,ADC_CONTR
                JNB     ACC.4,L_10bitADC_Loop1  ;等待 ADC 转换结束
                MOV     ADC_CONTR,#080H         ;清除标志
                MOV     A,ADC_RES
                MOV     B,#04H
                MUL     AB
                MOV     R7,A
                MOV     R6,B
                MOV     A,ADC_RESL
                ANL     A,#03H
                ORL     A,R7
                MOV     R7,A
                RET

F_AutoZero:                                     ;250ms 调用一次
                                                ;这是使用相邻 2 个采样的差的绝对值之和来检测。
                CLR     A
                MOV     R5,A
L_AutoZero_Loop:
                MOV     A,R5
                ADD     A,ACC
                ADD     A,#LOW (adc)
                MOV     R0,A
                MOV     A,@R0
                MOV     R6,A
                INC     R0
                MOV     A,@R0
                MOV     R7,A
                MOV     A,R5
                ADD     A,ACC
                ADD     A,#LOW (adc_prev+01H)
                MOV     R0,A
                CLR     C
                MOV     A,R7
                SUBB    A,@R0
                MOV     R3,A
                MOV     A,R6
                DEC     R0
                SUBB    A,@R0
                MOV     R2,A
                CLR     F0 ;按下
                JNB     ACC.7,L_AutoZero_1
                SETB    F0
                CLR     C
                CLR     A
                SUBB    A,R3
                MOV     R3,A
                MOV     A,R3
                CLR     A
                SUBB    A,R2
                MOV     R2,A
L_AutoZero_1:
                CLR     C                       ;计算 [R2 R3] - #20,if(k >= 20)
```

STC MCU

```
                MOV     A,R3
                SUBB    A,#20
                MOV     A,R2
                SUBB    A,#00H
                JC      L_AutoZero_2            ;[R2 R3] ,20,转
                MOV     A,#LOW (TouchZeroCnt)   ;如果变化比较大，则清 0 计数器 TouchZeroCnt[i] = 0;
                ADD     A,R5
                MOV     R0,A
                MOV     @R0,#0
                JNB     F0,L_AutoZero_3
                MOV     A,R5
                ADD     A,ACC
                ADD     A,#LOW (TouchZero)
                MOV     R0,A
                MOV     @R0,6
                INC     R0
                MOV     @R0,7
                SJMP    L_AutoZero_3
L_AutoZero_2:                                   ;变化比较小，则蠕动，自动 0 点跟踪
                                                ;连续检测到小变化 20 次/4 = 5 秒.
                MOV     A,#LOW (TouchZeroCnt)
                ADD     A,R5
                MOV     R0,A
                INC     @R0
                MOV     A,@R0
                CLR     C
                SUBB    A,#20
                JC      L_AutoZero_3            ;if(TouchZeroCnt[i] < 20),转
                MOV     @R0,#0                  ;TouchZeroCnt[i]      = 0;
                MOV     A,R5                    ;变化缓慢的值作为 0 点
                ADD     A,ACC
                ADD     A,#LOW (adc_prev)
                MOV     R0,A
                MOV     A,@R0
                MOV     R2,A
                INC     R0
                MOV     A,@R0
                MOV     R3,A
                MOV     A,R5
                ADD     A,ACC
                ADD     A,#LOW (TouchZero)
                MOV     R0,A
                MOV     @R0,2
                INC     R0
                MOV     @R0,3
L_AutoZero_3:                                   ;保存采样值 adc_prev[i] = j;
                MOV     A,R5
                ADD     A,ACC
                ADD     A,#LOW (adc_prev)
                MOV     R0,A
                MOV     @R0,6
                INC     R0
                MOV     @R0,7
                INC     R5
                MOV     A,R5
                XRL     A,#08H
                JZ      $ + 5H
                LJMP    L_AutoZero_Loop
                RET
```

```
F_check_adc:                                    ;判断键按下或释放,有回差控制
            MOV R4,7
            LCALL       F_Get_ADC10bitResult    ;返回的ADC 值在 [R6 R7]
            CLR         C
            MOV         A,#0FFH
            SUBB        A,R7
            MOV         R7,A
            MOV         A,#03H
            SUBB        A,R6
            MOV         R6,A
            MOV         A,R4                    ;保存 adc[index]
            ADD         A,ACC
            ADD         A,#LOW (adc)
            MOV         R0,A
            MOV         @R0,6
            INC         R0
            MOV         @R0,7
            MOV         A,R4
            ADD         A,ACC
            ADD         A,#LOW (TouchZero+01H)
            MOV         R1,A
            MOV         A,R4
            ADD         A,ACC
            ADD         A,#LOW (adc)
            MOV         R0,A
            MOV         A,@R0
            MOV         R6,A
            INC         R0
            MOV         A,@R0
            CLR         C
            SUBB        A,@R1                   ;计算 adc[index] - TouchZero[index]
            MOV         A,R6
            DEC         R1
            SUBB        A,@R1
            JNC         L_check_adc_1
            MOV         R7,#00H
            RET
L_check_adc_1:
            MOV         A,R4
            ADD         A,ACC
            ADD         A,#LOW (TouchZero+01H)
            MOV         R1,A
            MOV         A,R4
            ADD         A,ACC
            ADD         A,#LOW (adc+01H)
            MOV         R0,A
            CLR         C
            MOV         A,@R0
            SUBB        A,@R1
            MOV         R7,A
            DEC         R0
            MOV         A,@R0
            DEC         R1
            SUBB        A,@R1
            MOV         R6,A
            CLR         C
            MOV         A,R7
            SUBB        A,#40
```

```
                MOV     A,R6
                SUBB    A,#00H
                JC      L_check_adc_2           ;if(delta < 40),转
                MOV     R7,#1                   ;if(delta >= 40) return 1; //键按下 返回 1
                RET
L_check_adc_2:
                SETB    C
                MOV     A,R7
                SUBB    A,#20
                MOV     A,R6
                SUBB    A,#00H
                JNC     L_check_adc_3
                MOV     R7,#0
                RET
L_check_adc_3:
                MOV     R7,#2
                RET

F_ShowLED:
                MOV     R7,#0
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck0
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED0,C
L_QuitCheck0:
                MOV     R7,#1
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck1
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED1,C
L_QuitCheck1:
                MOV     R7,#2
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck2
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED2,C
L_QuitCheck2:
                MOV     R7,#3
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck3
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED3,C
L_QuitCheck3:
```

STC MCU

```
                MOV     R7,#4
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck4
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED4,C
L_QuitCheck4:
                MOV     R7,#5
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck5
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED5,C
L_QuitCheck5:
                MOV     R7,#6
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck6
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED6,C
L_QuitCheck6:
                MOV     R7,#7
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck7
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED7,C
L_QuitCheck7:
                RET

F_delay_ms:
                PUSH    3
                PUSH    4
L_delay_ms_1:
                MOV     R3,#HIGH (Fosc_KHZ / 13)
                MOV     R4,#LOW (Fosc_KHZ / 13)
L_delay_ms_2:
                MOV     A,R4
                DEC     R4
                JNZ     L_delay_ms_3
                DEC     R3
L_delay_ms_3:
                DEC     A
                ORL     A,R3
                JNZ     L_delay_ms_2
                DJNZ    R7,L_delay_ms_1
                POP     4
```

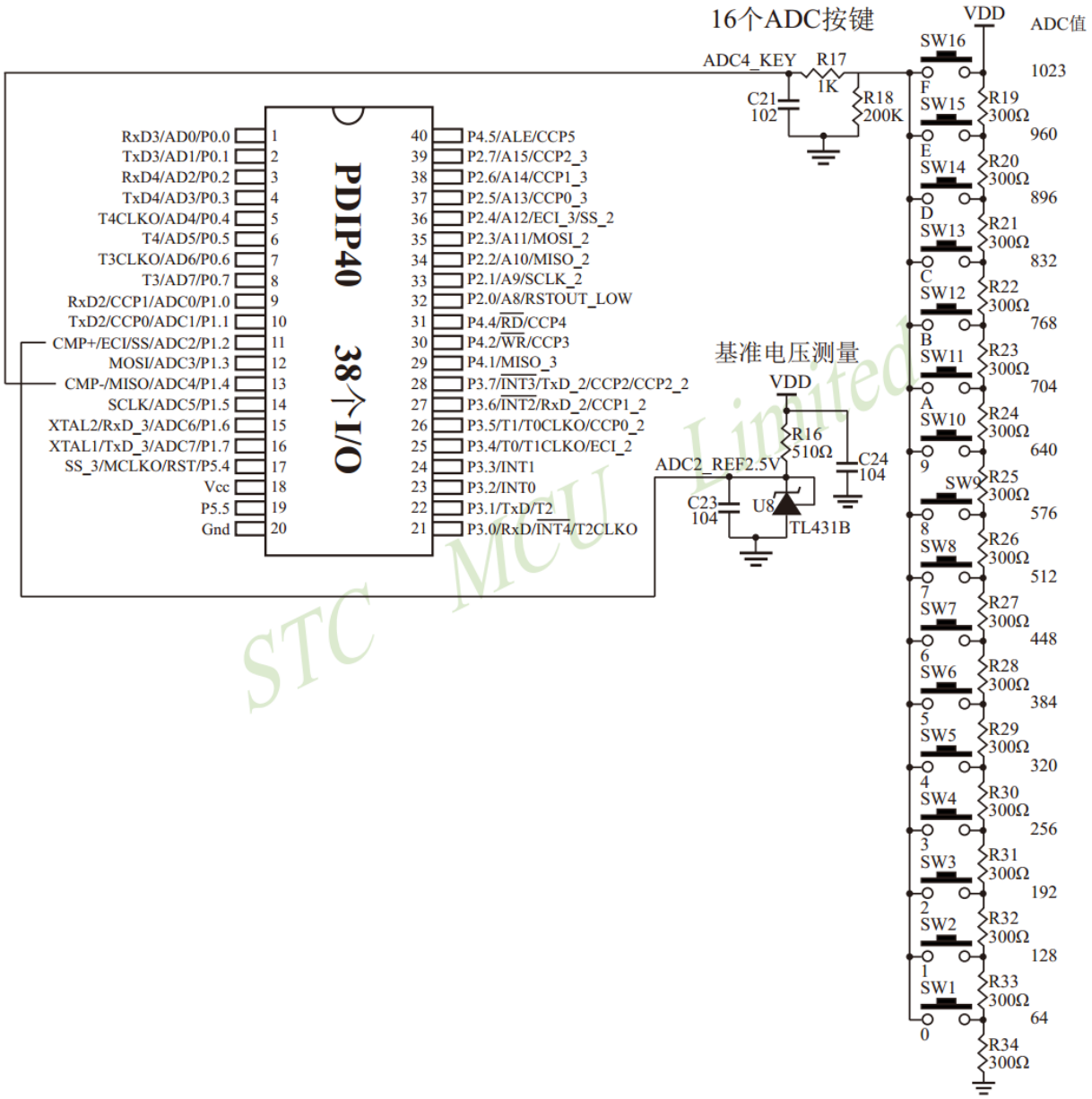
STC MCU

```
        POP     3
        RET

        END
```

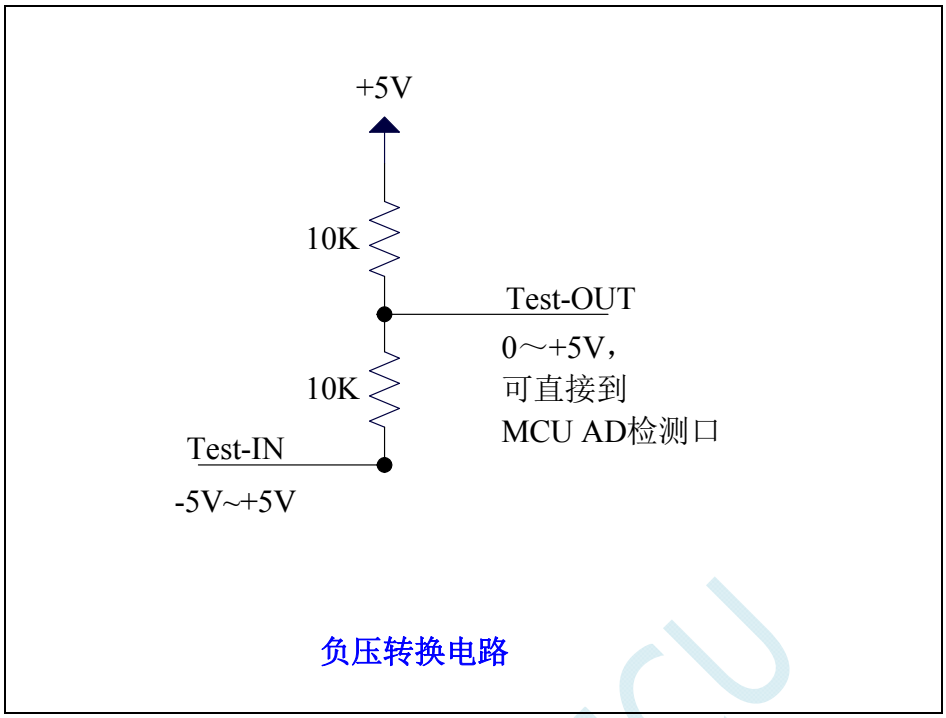
16.5.6 ADC作按键扫描应用线路图

读 ADC 键的方法：每隔 10ms 左右读一次 ADC 值，并且保存最后 3 次的读数，其变化比较小时再判断键。判断键有效时，允许一定的偏差，比如±16 个字的偏差。



STC MCU Limited

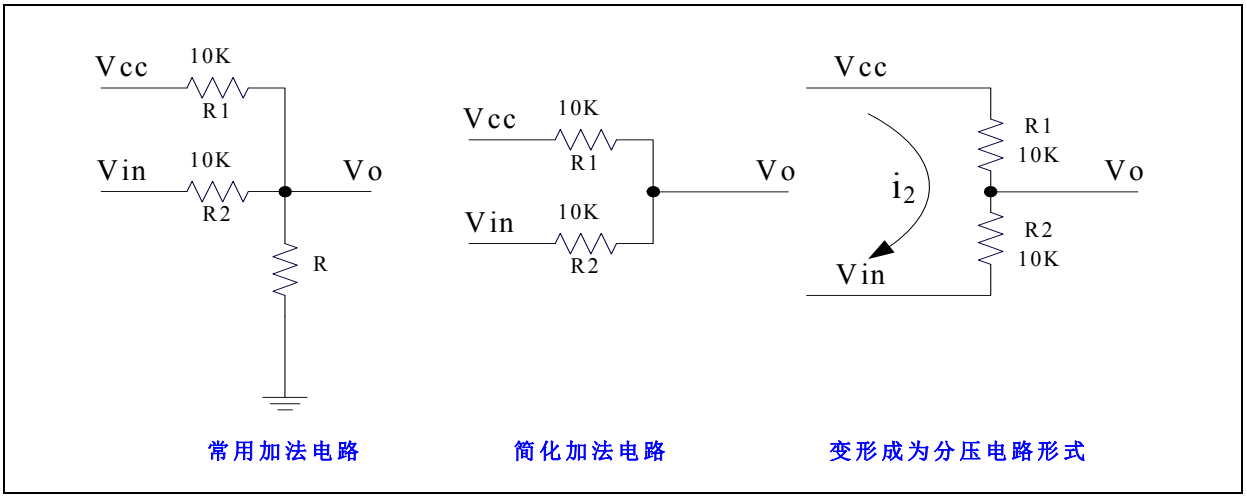
16.5.7 检测负电压参考线路图



负压转换电路

STC MCU

16.5.8 常用加法电路在ADC中的应用



参照分压电路得到公式 1

公式 1：Vo = Vin + i_2 * R2

公式 2：i_2 = (Vcc - Vin) / (R1 + R2)｛条件：流向 Vo 的电流 ≈0｝

将 R1=R2 代入公式 2 得公式 3

公式 3：i_2 = (Vcc - Vin) / 2R2

将公式 3 代入公式 1 得公式 4

公式 4：Vo = (Vcc + Vin) / 2

根据公式 4，可以将以上电路看成加法电路。

在单片机的模数转换测量中，要求被测电压大于 0 并且小于 VCC。如果被测电压小于 0V，可以利用加法电路将被测电压提升到 0V 以上。此时对被测电压的变化范围有一定的要求：

把上述条件代入公式 4 可得到下面 2 式

(Vcc + Vin) / 2 > 0　　即 Vin > -Vcc

(Vcc + Vin) / 2 < Vcc　　即 Vin < Vcc

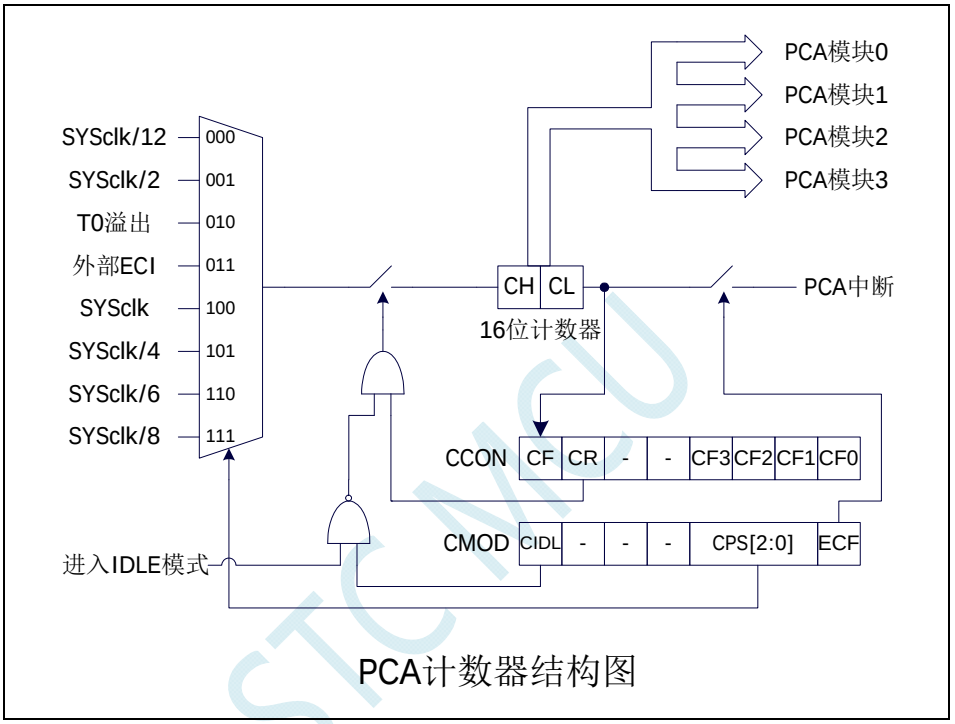
上面 2 式可以合起来：　　**-Vcc < Vin < Vcc**

17 PCA/CCP/PWM应用

特别注意：当使能 PCA 模块的的高速脉冲输出或者使能 PWM 输出时，相应的输出脚可设置为准双向口模式或者推挽输出模式，但发送脚一定不能设置开漏模式，否则没波形输出

STC8A/8F 系列单片机内部集成了 4 组可编程计数器阵列（PCA/CCP/PWM）模块，可用于软件定时器、外部脉冲捕获、高速脉冲输出和 PWM 脉宽调制输出。

PCA 内部含有一个特殊的 16 位计数器，3 组 PCA 模块均与之相连接。PCA 计数器的结构图如下：



PCA计数器结构图

17.1 PCA相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CCON	PCA 控制寄存器	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0	00xx,x000
CMOD	PCA 模式寄存器	D9H	CIDL	-	-	-	CPS[2:0]			ECF	0xxx,0000
CCAPM0	PCA 模块 0 模式控制寄存器	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA 模块 1 模式控制寄存器	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA 模块 2 模式控制寄存器	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
CCAPM3	PCA 模块 3 模式控制寄存器	DDH	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3	x000,0000
CL	PCA 计数器低字节	E9H									0000,0000
CCAP0L	PCA 模块 0 低字节	EAH									0000,0000
CCAP1L	PCA 模块 1 低字节	EBH									0000,0000
CCAP2L	PCA 模块 2 低字节	ECH									0000,0000
CCAP3L	PCA 模块 3 低字节	EDH									0000,0000
PCA_PWM0	PCA0 的 PWM 模式寄存器	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L	0000,0000
PCA_PWM1	PCA1 的 PWM 模式寄存器	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L	0000,0000

PCA_PWM2	PCA2 的 PWM 模式寄存器	F4H	EBS2[1:0]	XCCAP2H[1:0]	XCCAP2L[1:0]	EPC2H	EPC2L	0000,0000
PCA_PWM3	PCA3 的 PWM 模式寄存器	F5H	EBS3[1:0]	XCCAP3H[1:0]	XCCAP3L[1:0]	EPC3H	EPC3L	0000,0000
CH	PCA 计数器高字节	F9H						0000,0000
CCAP0H	PCA 模块 0 高字节	FAH						0000,0000
CCAP1H	PCA 模块 1 高字节	FBH						0000,0000
CCAP2H	PCA 模块 2 高字节	FCH						0000,0000
CCAP3H	PCA 模块 3 高字节	FDH						0000,0000

17.1.1 PCA控制寄存器（CCON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCON	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0

CF：PCA 计数器溢出中断标志。当 PCA 的 16 位计数器计数发生溢出时，硬件自动将此位置 1，并向 CPU 提出中断请求。此标志位需要软件清零。

CR：PCA 计数器允许控制位。

0：停止 PCA 计数

1：启动 PCA 计数

CCFn（n=0,1,2,3）：PCA 模块中断标志。当 PCA 模块发生匹配或者捕获时，硬件自动将此位置 1，并向 CPU 提出中断请求。此标志位需要软件清零。

17.1.2 PCA模式寄存器（CMOD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	CIDL	-	-	-	CPS[2:0]			ECF

CIDL：空闲模式下是否停止 PCA 计数。

0：空闲模式下 PCA 继续计数

1：空闲模式下 PCA 停止计数

CPS[2:0]：PCA 计数脉冲源选择位

CPS[2:0]	PCA 的输入时钟源
000	系统时钟/12
001	系统时钟/2
010	定时器 0 的溢出脉冲
011	ECI 脚的外部输入时钟
100	系统时钟
101	系统时钟/4
110	系统时钟/6
111	系统时钟 8

ECF：PCA 计数器溢出中断允许位。

0：禁止 PCA 计数器溢出中断

1：使能 PCA 计数器溢出中断

17.1.3 PCA计数器寄存器（CL，CH）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CL	E9H								

CH	F9H	

由 CL 和 CH 两个字节组合成一个 16 位计数器，CL 为低 8 位计数器，CH 为高 8 位计数器。每个 PCA 时钟 16 位计数器自动加 1。

17.1.4 PCA模块模式控制寄存器（CCAPMn）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCAPM0	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0
CCAPM1	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1
CCAPM2	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2
CCAPM3	DDH	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3

ECOMn：允许 PCA 模块 n 的比较功能
CCAPPn：允许 PCA 模块 n 进行上升沿捕获
CCAPNn：允许 PCA 模块 n 进行下降沿捕获
MATn：允许 PCA 模块 n 的匹配功能
TOGn：允许 PCA 模块 n 的高速脉冲输出功能
PWMn：允许 PCA 模块 n 的脉宽调制输出功能
ECCFn：允许 PCA 模块 n 的匹配/捕获中断

17.1.5 PCA模块模式捕获值/比较值寄存器（CCAPnL，CCAPnH）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCAP0L	EAH								
CCAP1L	EBH								
CCAP2L	ECH								
CCAP3L	EDH								
CCAP0H	FAH								
CCAP1H	FBH								
CCAP2H	FCH								
CCAP3H	FDH								

当 PCA 模块捕获功能使能时，CCAPnL 和 CCAPnH 用于保存发生捕获时的 PCA 的计数值（CL 和 CH）；当 PCA 模块比较功能使能时，PCA 控制器会将当前 CL 和 CH 中的计数值与保存在 CCAPnL 和 CCAPnH 中的值进行比较，并给出比较结果；当 PCA 模块匹配功能使能时，PCA 控制器会将当前 CL 和 CH 中的计数值与保存在 CCAPnL 和 CCAPnH 中的值进行比较，看是否匹配（相等），并给出匹配结果。

17.1.6 PCA模块PWM模式控制寄存器（PCA_PWMn）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCA_PWM0	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L
PCA_PWM1	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L
PCA_PWM2	F4H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L
PCA_PWM3	F5H	EBS3[1:0]		XCCAP3H[1:0]		XCCAP3L[1:0]		EPC3H	EPC3L

EBSn[1:0]：PCA 模块 n 的 PWM 位数控制

EBSn[1:0]	PWM 位数	重载值	比较值

00	8 位 PWM	{EPCnH, CCAPnH[7:0]}	{EPCnL, CCAPnL[7:0]}
01	7 位 PWM	{EPCnH, CCAPnH[6:0]}	{EPCnL, CCAPnL[6:0]}
10	6 位 PWM	{EPCnH, CCAPnH[5:0]}	{EPCnL, CCAPnL[5:0]}
11	10 位 PWM	{EPCnH, XCCAPnH[1:0], CCAPnH[7:0]}	{EPCnL, XCCAPnL[1:0], CCAPnL[7:0]}

XCCAPnH[1:0]：10 位 PWM 的第 9 位和第 10 位的重载值

XCCAPnL[1:0]：10 位 PWM 的第 9 位和第 10 位的比较值

EPCnH：PWM 模式下，重载值的最高位（8 位 PWM 的第 9 位，7 位 PWM 的第 8 位，6 位 PWM 的第 7 位，10 位 PWM 的第 11 位）

EPCnL：PWM 模式下，比较值的最高位（8 位 PWM 的第 9 位，7 位 PWM 的第 8 位，6 位 PWM 的第 7 位，10 位 PWM 的第 11 位）

注意：在更新 10 位 PWM 的重载值时，必须先写高两位 XCCAPnH[1:0]，再写低 8 位 CCAPnH[7:0]。

STC MCU

17.2 PCA工作模式

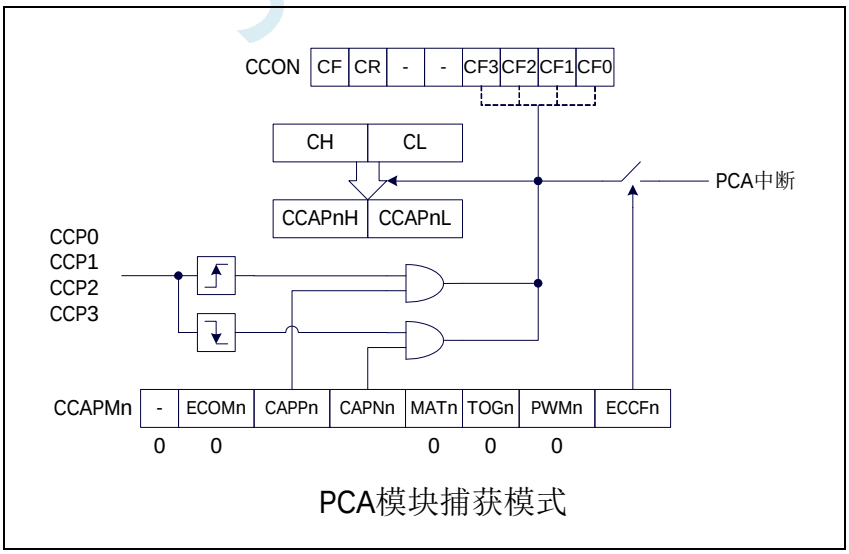
STCA/8F 系列单片机共有 4 组 PCA 模块，每组模块都可独立设置工作模式。模式设置如下所示：

CCAPMn								模块功能
-	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	
-	0	0	0	0	0	0	0	无操作
-	1	0	0	0	0	1	0	6/7/8/10 位 PWM 模式，无中断
-	1	1	0	0	0	1	1	6/7/8/10 位 PWM 模式，产生上升沿中断
-	1	0	1	0	0	1	1	6/7/8/10 位 PWM 模式，产生下降沿中断
-	1	1	1	0	0	1	1	6/7/8/10 位 PWM 模式，产生边沿中断
-	0	1	0	0	0	0	x	16 位上升沿捕获
-	0	0	1	0	0	0	x	16 位下降沿捕获
-	0	1	1	0	0	0	x	16 位边沿捕获
-	1	0	0	1	0	0	x	16 位软件定时器
-	1	0	0	1	1	0	x	16 位高速脉冲输出

17.2.1 捕获模式

要使一个 PCA 模块工作在捕获模式，寄存器 CCAPMn 中的 CAPNn 和 CAPPn 至少有一位必须置 1（也可两位都置 1）。PCA 模块工作于捕获模式时，对模块的外部 CCP0/CCP1/CCP2 管脚的输入跳变进行采样。当采样到有效跳变时，PCA 控制器立即将 PCA 计数器 CH 和 CL 中的计数值装载到模块的捕获寄存器中 CCAPnL 和 CCAPnH，同时将 CCON 寄存器中相应的 CCFn 置 1。若 CCAPMn 中的 ECCFn 位被设置为 1，将产生中断。由于所有 PCA 模块的中断入口地址是共享的，所以在中断服务程序中需要判断是哪一个模块产生了中断，并注意中断标志位需要软件清零。

PCA 模块工作于捕获模式的结构图如下图所示：



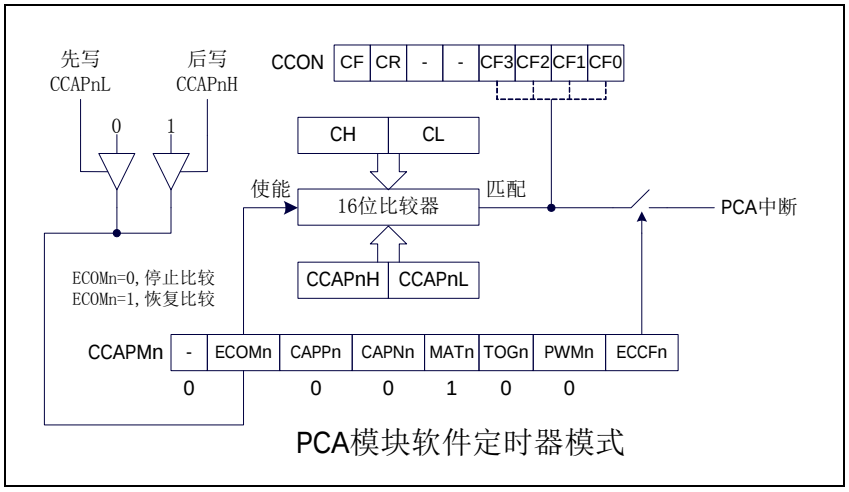
PCA模块捕获模式

17.2.2 软件定时器模式

通过置位 CCAPMn 寄存器的 ECOM 和 MAT 位，可使 PCA 模块用作软件定时器。PCA 计数器值 CL 和 CH 与模块捕获寄存器的值 CCAPnL 和 CCAPnH 相比较，当两者相等时，CCON 中的 CCFn 会被

置 1，若 CCAPMn 中的 ECCFn 被设置为 1 时将产生中断。CCFn 标志位需要软件清零。

PCA 模块工作于软件定时器模式的结构图如下图所示：

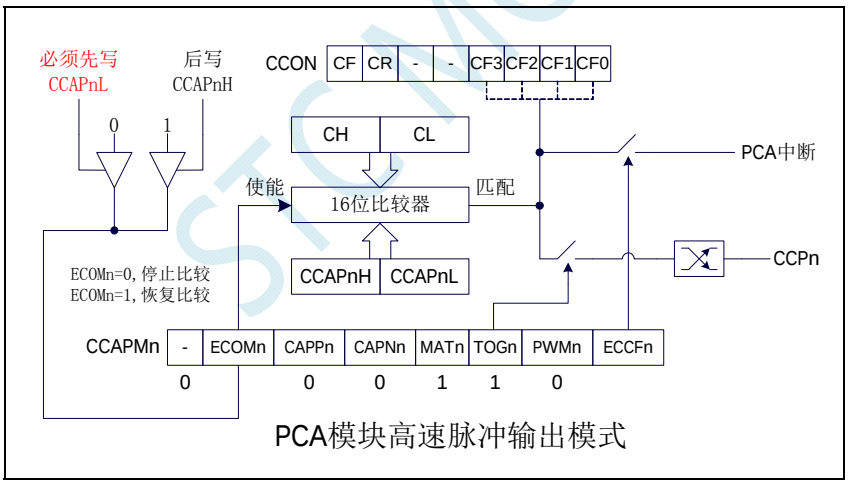


PCA模块软件定时器模式

17.2.3 高速脉冲输出模式

当 PCA 计数器的计数值与模块捕获寄存器的值相匹配时,PCA 模块的 CCPn 输出将发生翻转转。要激活高速脉冲输出模式，CCAPMn 寄存器的 TOGn、MATn 和 ECOMn 位必须都置 1。

PCA 模块工作于高速脉冲输出模式的结构图如下图所示：



PCA模块高速脉冲输出模式

17.2.4 PWM脉宽调制模式及频率计算公式

17.2.4.1 8 位PWM模式

脉宽调制是使用程序来控制波形的占空比、周期、相位波形的一种技术，在三相电机驱动、D/A 转换等场合有广泛的应用。STC8 系列单片机的 PCA 模块可以通过设定各自的 PCA_PWMn 寄存器使其工作于 8 位 PWM 或 7 位 PWM 或 6 位 PWM 或 10 位 PWM 模式。要使能 PCA 模块的 PWM 功能，模块寄存器 CCAPMn 的 PWMn 和 ECOMn 位必须置 1。

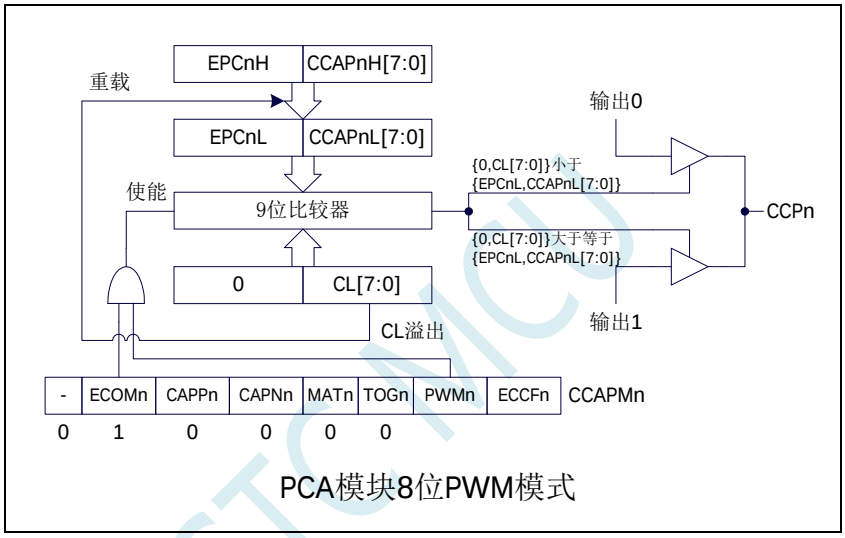
PCA_PWMn 寄存器中的 EBSn[1:0]设置为 00 时，PCA 模块 n 工作于 8 位 PWM 模式，此时将{0,CL[7:0]}与捕获寄存器{EPCnL,CCAPnL[7:0]}进行比较。当 PCA 模块工作于 8 位 PWM 模式时，由于所有模块共

用一个PCA计数器，所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,CCAPnL[7:0]}进行设置。当{0,CL[7:0]}的值小于{EPCnL,CCAPnL[7:0]}时，输出为低电平；当{0,CL[7:0]}的值等于或大于{EPCnL,CCAPnL[7:0]}时，输出为高电平。当CL[7:0]的值由FF变为00溢出时，{EPCnH,CCAPnH[7:0]}的内容重新装载到{EPCnL,CCAPnL[7:0]}中。这样就可实现无干扰地更新 PWM。

$$8位模式的PWM频率 = \frac{PCA时钟输入源频率}{256}$$

当EPCnH=0及CCAPnH=00H时，PWM固定输出高
当EPCnH=1及CCAPnH=FFH时，PWM固定输出低

PCA 模块工作于 8 位 PWM 模式的结构图如下图所示：



PCA模块8位PWM模式

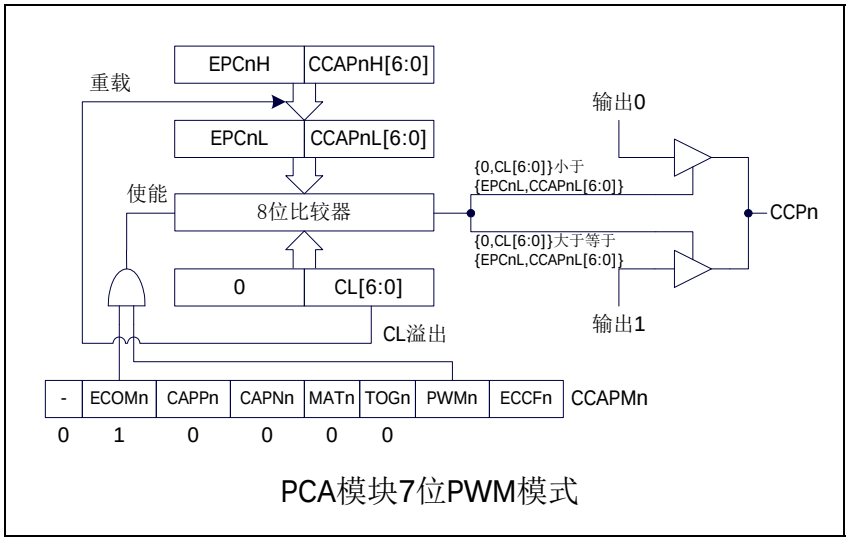
17.2.4.2 7 位PWM模式

PCA_PWMn 寄存器中的EBSn[1:0]设置为01时，PCA 模块n工作于7位PWM模式，此时将{0,CL[6:0]}与捕获寄存器{EPCnL,CCAPnL[6:0]}进行比较。当 PCA 模块工作于 7 位 PWM 模式时，由于所有模块共用一个PCA计数器，所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,CCAPnL[6:0]}进行设置。当{0,CL[6:0]}的值小于{EPCnL,CCAPnL[6:0]}时，输出为低电平；当{0,CL[6:0]}的值等于或大于{EPCnL,CCAPnL[6:0]}时，输出为高电平。当CL[6:0]的值由7F变为00溢出时，{EPCnH,CCAPnH[6:0]}的内容重新装载到{EPCnL,CCAPnL[6:0]}中。这样就可实现无干扰地更新 PWM。

$$7位模式的PWM频率 = \frac{PCA时钟输入源频率}{128}$$

当EPCnH=0及CCAPnH=00H时，PWM固定输出高
当EPCnH=1及CCAPnH=FFH时，PWM固定输出低

PCA 模块工作于 7 位 PWM 模式的结构图如下图所示：



PCA模块7位PWM模式

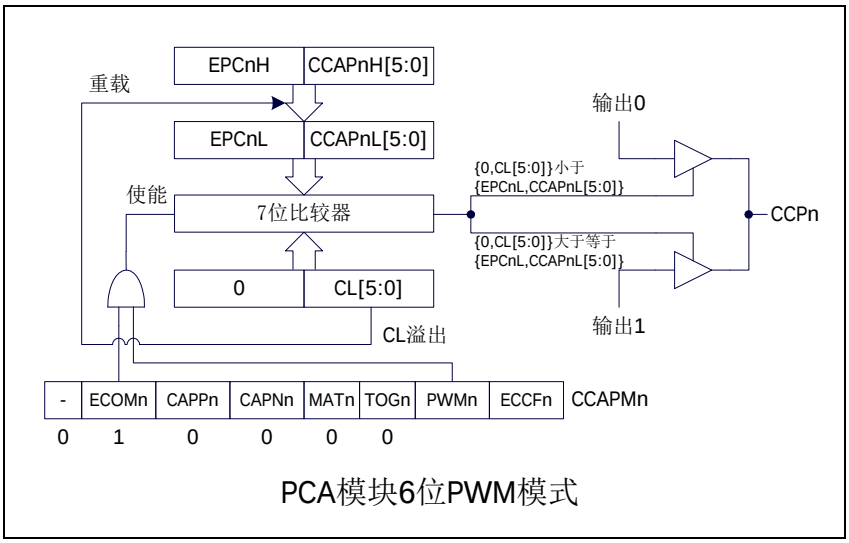
17.2.4.3 6 位PWM模式

PCA_PWMn 寄存器中的 EBSn[1:0]设置为 10 时，PCA 模块 n 工作于 6 位 PWM 模式，此时将{0,CL[5:0]}与捕获寄存器{EPCnL,CCAPnL[5:0]}进行比较。当 PCA 模块工作于 6 位 PWM 模式时，由于所有模块共用一个 PCA 计数器，所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,CCAPnL[5:0]}进行设置。当{0,CL[5:0]}的值小于{EPCnL,CCAPnL[5:0]}时，输出为低电平；当{0,CL[5:0]}的值等于或大于{EPCnL,CCAPnL[5:0]}时，输出为高电平。当 CL[5:0]的值由 3F 变为 00 溢出时，{EPCnH,CCAPnH[5:0]}的内容重新装载到{EPCnL,CCAPnL[5:0]}中。这样就可实现无干扰地更新 PWM。

$$8\text{位模式的PWM频率} = \frac{\text{PCA时钟输入源频率}}{64}$$

当EPCnH=0及CCAPnH=00H时，PWM固定输出高
当EPCnH=1及CCAPnH=FFH时，PWM固定输出低

PCA 模块工作于 6 位 PWM 模式的结构图如下图所示：



PCA模块6位PWM模式

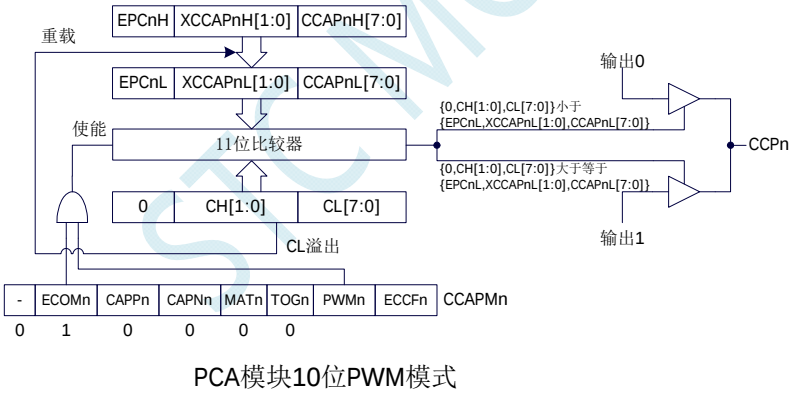
17.2.4.4 10 位PWM模式

PCA_PWMn 寄存器中的 EBSn[1:0]设置为 11 时，PCA 模块 n 工作于 10 位 PWM 模式，此时将{CH[1:0],CL[7:0]}与捕获寄存器{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}进行比较。当 PCA 模块工作于 10 位 PWM 模式时，由于所有模块共用一个 PCA 计数器，所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}进行设置。当{CH[1:0],CL[7:0]}的值小于{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}时，输出为低电平；当{CH[1:0],CL[7:0]}的值等于或大于{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}时，输出为高电平。当{CH[1:0],CL[7:0]}的值由 3FF 变为 00 溢出时，{EPCnH,XCCAPnH[1:0],CCAPnH[7:0]}的内容重新装载到{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}中。这样就可实现无干扰地更新 PWM。

$$10\text{位模式的PWM频率} = \frac{\text{PCA时钟输入源频率}}{1024}$$

当EPCnH=0,XCCAP0H=0及CCAPnH=C0H时，PWM固定输出高
当EPCnL=1,XCCAP0H=3及CCAPnH=FFH时，PWM固定输出低

PCA 模块工作于 10 位 PWM 模式的结构图如下图所示：



PCA模块10位PWM模式

17.2.4.5 如何控制PWM固定输出高电平/低电平

当 PCA_PWMn &= 0xC0，CCAPnH = 0x00 时，PWM 固定输出高电平
当 PCA_PWMn |= 0x3F，CCAPnH = 0xFF 时，PWM 固定输出低电平

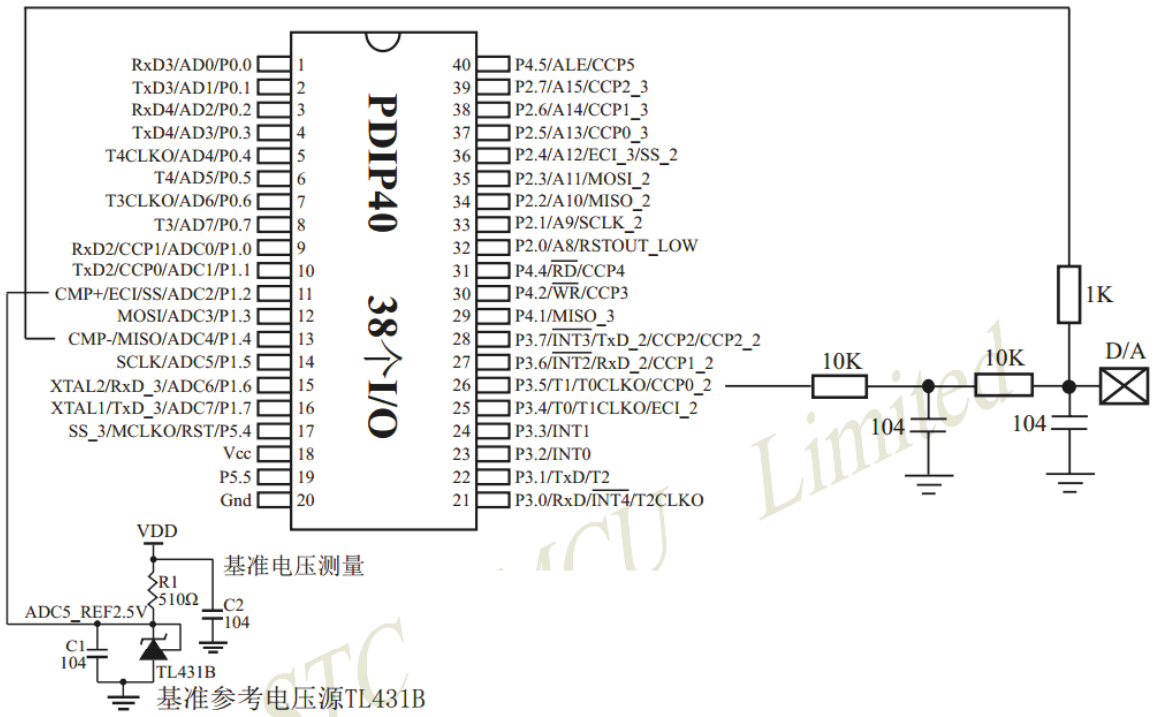
注意事项：

1. **4 通道 PCA 工作于 PWM 时，切换到 P1.4~P1.7 或 P2.3~P2.6 或 P3.3~P3.0，，情况同上类似，IO 设置为准双向口或推挽输出，PWM 输出正常，均为推挽输出，并且再直接操作 IO 将不影响 PWM 波形。但是如果将 IO 设置为开漏输出（允许内部上拉电阻或外接上拉电阻），则如果对应的 IO 输出高电平，PWM 无输出（IO 为高阻），而对应的 IO 输出低电平，PWM 有推挽输出。**

2. **4 通道 PCA 工作于高速输出时，受到同一个端口其余 IO 翻转的影响，比如 PCA3 从 P2.6 高速输出脉冲，P2 口除了 P2.6 外任何一个 P2 口的改变都会影响到 P2.6 波形，哪怕是将 IO 设置为高阻。**

STC MCU

17.3 利用CCP/PCA/PWM模块实现 8~16 位DAC的参考线路图



如应用简单，可无需基准参考电压源，直接与Vcc比较即可。

利用CCP/PCA模块的高速脉冲输出功能实现9~16位PWM来实现9~16位DAC，或用本身的硬件8位PWM来实现8位DAC，单片机本身也有10位ADC。

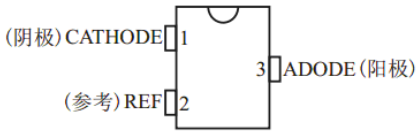
提示：

（1）PWM频率越高，输出波形越平滑。

（2）如果工作电压为5V，需输出1V电压，则设置高电平为1/5，低电平为4/5，则PWM输出电压就为1V。

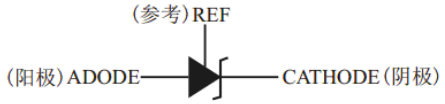
（3）如果要输出高精准电压，建议用A/D检测输出的电压值，然后根据A/D检测的电压值逐步调整到所需要的电压。

基准参考电压源TL431B



SOT23-3封装，RMB￥0.15~0.3

基准参考电压源TL431B的符号



如应用简单，可无需基准参考电压源，直接与Vcc比较即可。

17.4 范例程序

17.4.1 PCA输出PWM（6/7/8/10 位）

汇编代码

```
;测试工作频率为11.0592MHz

CCON    DATA    0D8H
CF      BIT     CCON.7
CR      BIT     CCON.6
CCF3    BIT     CCON.3
CCF2    BIT     CCON.2
CCF1    BIT     CCON.1
CCF0    BIT     CCON.0
CMOD    DATA    0D9H
CL      DATA    0E9H
CH      DATA    0F9H
CCAPM0  DATA    0DAH
CCAP0L  DATA    0EAH
CCAP0H  DATA    0FAH
PCA_PWM0 DATA   0F2H
CCAPM1  DATA    0DBH
CCAP1L  DATA    0EBH
CCAP1H  DATA    0FBH
PCA_PWM1 DATA   0F3H
CCAPM2  DATA    0DCH
CCAP2L  DATA    0ECH
CCAP2H  DATA    0FCH
PCA_PWM2 DATA   0F4H
CCAPM3  DATA    0DDH
CCAP3L  DATA    0EDH
CCAP3H  DATA    0FDH
PCA_PWM3 DATA   0F5H

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP,#3FH

        MOV     CCON,#00H
        MOV     CMOD,#08H       ;PCA 时钟为系统时钟
        MOV     CL,#00H
        MOV     CH,#0H
        MOV     CCAPM0,#42H     ;PCA 模块 0 为 PWM 工作模式
        MOV     PCA_PWM0,#80H   ;PCA 模块 0 输出 6 位 PWM
        MOV     CCAP0L,#20H     ;PWM 占空比为 50%[(40H-20H)/40H]
        MOV     CCAP0H,#20H
        MOV     CCAPM1,#42H     ;PCA 模块 1 为 PWM 工作模式
        MOV     PCA_PWM1,#40H   ;PCA 模块 1 输出 7 位 PWM
        MOV     CCAP1L,#20H     ;PWM 占空比为 75%[(80H-20H)/80H]
        MOV     CCAP1H,#20H
        MOV     CCAPM2,#42H     ;PCA 模块 2 为 PWM 工作模式
        MOV     PCA_PWM2,#00H   ;PCA 模块 2 输出 8 位 PWM
        MOV     CCAP2L,#20H     ;PWM 占空比为 87.5%[(100H-20H)/100H]
```

```
        MOV     CCAP2H,#20H
        MOV     CCAPM3,#42H             ;PCA 模块 3 为 PWM 工作模式
        MOV     PCA_PWM3,#0C0H          ;PCA 模块 3 输出 10 位 PWM
        MOV     CCAP3L,#20H             ;PWM 占空比为 96.875%[(400H-20H)/400H]
        MOV     CCAP3H,#20H
        SETB    CR                      ;启动 PCA 计时器

        JMP     $

        END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为 11.0592MHz

sfr     CCON    =   0xd8;
sbit    CF      =   CCON^7;
sbit    CR      =   CCON^6;
sbit    CCF3    =   CCON^3;
sbit    CCF2    =   CCON^2;
sbit    CCF1    =   CCON^1;
sbit    CCF0    =   CCON^0;
sfr     CMOD    =   0xd9;
sfr     CL      =   0xe9;
sfr     CH      =   0xf9;
sfr     CCAPM0  =   0xda;
sfr     CCAP0L  =   0xea;
sfr     CCAP0H  =   0xfa;
sfr     PCA_PWM0=   0xf2;
sfr     CCAPM1  =   0xdb;
sfr     CCAP1L  =   0xeb;
sfr     CCAP1H  =   0xfb;
sfr     PCA_PWM1=   0xf3;
sfr     CCAPM2  =   0xdc;
sfr     CCAP2L  =   0xec;
sfr     CCAP2H  =   0xfc;
sfr     PCA_PWM2=   0xf4;
sfr     CCAPM3  =   0xdd;
sfr     CCAP3L  =   0xed;
sfr     CCAP3H  =   0xfd;
sfr     PCA_PWM3=   0xf5;

void main()
{
    CCON = 0x00;
    CMOD = 0x08;                //PCA 时钟为系统时钟
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x42;              //PCA 模块 0 为 PWM 工作模式
    PCA_PWM0 = 0x80;            //PCA 模块 0 输出 6 位 PWM
    CCAP0L = 0x20;              //PWM 占空比为 50%[(40H-20H)/40H]
    CCAP0H = 0x20;
    CCAPM1 = 0x42;              //PCA 模块 1 为 PWM 工作模式
    PCA_PWM1 = 0x40;            //PCA 模块 1 输出 7 位 PWM
    CCAP1L = 0x20;              //PWM 占空比为 75%[(80H-20H)/80H]
    CCAP1H = 0x20;
```

STC MCU

```
    CCAPM2 = 0x42;          //PCA 模块 2 为 PWM 工作模式
    PCA_PWM2 = 0x00;        //PCA 模块 2 输出 8 位 PWM
    CCAP2L = 0x20;          //PWM 占空比为 87.5%[(100H-20H)/100H]
    CCAP2H = 0x20;
    CCAPM3 = 0x42;          //PCA 模块 3 为 PWM 工作模式
    PCA_PWM3 = 0xc0;        //PCA 模块 3 输出 10 位 PWM
    CCAP3L = 0x20;          //PWM 占空比为 96.875%[(400H-20H)/400H]
    CCAP3H = 0x20;
    CR = 1;                 //启动 PCA 计时器

    while (1);
}
```

17.4.2 PCA捕获测量脉冲宽度

汇编代码

```
;测试工作频率为 11.0592MHz

CCON        DATA    0D8H
CF          BIT     CCON.7
CR          BIT     CCON.6
CCF3        BIT     CCON.3
CCF2        BIT     CCON.2
CCF1        BIT     CCON.1
CCF0        BIT     CCON.0
CMOD        DATA    0D9H
CL          DATA    0E9H
CH          DATA    0F9H
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H
CCAPM3      DATA    0DDH
CCAP3L      DATA    0EDH
CCAP3H      DATA    0FDH
PCA_PWM3    DATA    0F5H

CNT         DATA    20H
COUNT0      DATA    21H             ;3 bytes
COUNT1      DATA    24H             ;3 bytes
LENGTH      DATA    27H             ;3 bytes, (COUNT1-COUNT0)

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            PUSH    ACC
```

```
                PUSH    PSW
                JNB     CF,CHECKCCF0
                CLR     CF              ;清中断标志
                INC     CNT             ; PCA 计时溢出次数+1
CHECKCCF0:
                JNB     CCF0,ISREXIT
                CLR     CCF0
                MOV     COUNT0,COUNT1   ;备份上一次的捕获值
                MOV     COUNT0+1,COUNT1+1
                MOV     COUNT0+2,COUNT1+2
                MOV     COUNT1,CNT      ;保存本次的捕获值
                MOV     COUNT1+1,CCAP0H
                MOV     COUNT1+2,CCAP0L
                CLR     C               ;计算两次的捕获差值
                MOV     A,COUNT1+2
                SUBB    A,COUNT0+2
                MOV     LENGTH+2,A
                MOV     A,COUNT1+1
                SUBB    A,COUNT0+1
                MOV     LENGTH+1,A
                MOV     A,COUNT1
                SUBB    A,COUNT0
                MOV     LENGTH,A        ;LENGTH 保存的即为捕获的脉冲宽度
ISREXIT:
                POP     PSW
                POP     ACC
                RETI

MAIN:
                MOV     SP,#3FH

                CLR     A
                MOV     CNT,A           ;用户变量初始化
                MOV     COUNT0,A
                MOV     COUNT0+1,A
                MOV     COUNT0+2,A
                MOV     COUNT1,A
                MOV     COUNT1+1,A
                MOV     COUNT1+2,A
                MOV     LENGTH,A
                MOV     LENGTH+1,A
                MOV     LENGTH+2,A

                MOV     CCON,#00H
                MOV     CMOD,#09H       ;PCA 时钟为系统时钟,使能 PCA 计时中断
                MOV     CL,#00H
                MOV     CH,#0H
                MOV     CCAPM0,#11H     ;PCA 模块 0 为 16 位捕获模式（下降沿捕获）
;               MOV     CCAPM0,#21H     ;PCA 模块 0 为 16 位捕获模式（上升沿捕获）
;               MOV     CCAPM0,#31H     ;PCA 模块 0 为 16 位捕获模式（边沿捕获）
                MOV     CCAP0L,#00H
                MOV     CCAP0H,#00H
                SETB    CR              ;启动 PCA 计时器
                SETB    EA

                JMP     $

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为 11.0592MHz

sfr         CCON        =   0xd8;
sbit        CF          =   CCON^7;
sbit        CR          =   CCON^6;
sbit        CCF3        =   CCON^3;
sbit        CCF2        =   CCON^2;
sbit        CCF1        =   CCON^1;
sbit        CCF0        =   CCON^0;
sfr         CMOD        =   0xd9;
sfr         CL          =   0xe9;
sfr         CH          =   0xf9;
sfr         CCAPM0      =   0xda;
sfr         CCAP0L      =   0xea;
sfr         CCAP0H      =   0xfa;
sfr         PCA_PWM0    =   0xf2;
sfr         CCAPM1      =   0xdb;
sfr         CCAP1L      =   0xeb;
sfr         CCAP1H      =   0xfb;
sfr         PCA_PWM1    =   0xf3;
sfr         CCAPM2      =   0xdc;
sfr         CCAP2L      =   0xec;
sfr         CCAP2H      =   0xfc;
sfr         PCA_PWM2    =   0xf4;
sfr         CCAPM3      =   0xdd;
sfr         CCAP3L      =   0xed;
sfr         CCAP3H      =   0xfd;
sfr         PCA_PWM3    =   0xf5;

unsigned char   cnt;                    //存储 PCA 计时溢出次数
unsigned long   count0;                 //记录上一次的捕获值
unsigned long   count1;                 //记录本次的捕获值
unsigned long   length;                 //存储信号的时间长度

void PCA_Isr() interrupt 7
{
    if (CF)
    {
        CF = 0;
        cnt++;                          //PCA 计时溢出次数+1
    }
    if (CCF0)
    {
        CCF0 = 0;
        count0 = count1;                //备份上一次的捕获值
        ((unsigned char *)&count1)[3] = CCAP0L;
        ((unsigned char *)&count1)[2] = CCAP0H;
        ((unsigned char *)&count1)[1] = cnt;
        ((unsigned char *)&count1)[0] = 0;
        length = count1 - count0;       //length 保存的即为捕获的脉冲宽度
    }
}

void main()
{
```

```
    cnt = 0;                        //用户变量初始化
    count0 = 0;
    count1 = 0;
    length = 0;
    CCON = 0x00;
    CMOD = 0x09;                    //PCA 时钟为系统时钟,使能 PCA 计时中断
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x11;                  //PCA 模块 0 为 16 位捕获模式（下降沿捕获）
    CCAPM0 = 0x21;                  //PCA 模块 0 为 16 位捕获模式（下降沿捕获）
    CCAPM0 = 0x31;                  //PCA 模块 0 为 16 位捕获模式（下降沿捕获）
    CCAP0L = 0x00;
    CCAP0H = 0x00;
    CR = 1;                         //启动 PCA 计时器
    EA = 1;

    while (1);
}
```

17.4.3 PCA实现 16 位软件定时

汇编代码

```
;测试工作频率为 11.0592MHz

CCON        DATA    0D8H
CF          BIT     CCON.7
CR          BIT     CCON.6
CCF3        BIT     CCON.3
CCF2        BIT     CCON.2
CCF1        BIT     CCON.1
CCF0        BIT     CCON.0
CMOD        DATA    0D9H
CL          DATA    0E9H
CH          DATA    0F9H
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H
CCAPM3      DATA    0DDH
CCAP3L      DATA    0EDH
CCAP3H      DATA    0FDH
PCA_PWM3    DATA    0F5H

T50HZ       EQU     2400H               ;11059200/12/2/50

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR
```

```
                ORG     0100H
PCAISR:
                PUSH    ACC
                PUSH    PSW
                CLR     CCF0
                MOV     A,CCAP0L
                ADD     A,#LOW T50HZ
                MOV     CCAP0L,A
                MOV     A,CCAP0H
                ADDC    A,#HIGH T50HZ
                MOV     CCAP0H,A
                CPL     P1.0                ;测试端口,闪烁频率为50Hz
                POP     PSW
                POP     ACC
                RETI

MAIN:
                MOV     SP,#3FH

                MOV     CCON,#00H
                MOV     CMOD,#00H           ;PCA 时钟为系统时钟/12
                MOV     CL,#00H
                MOV     CH,#0H
                MOV     CCAPM0,#49H         ;PCA 模块0 为16 位定时器模式
                MOV     CCAP0L,#LOW T50HZ
                MOV     CCAP0H,#HIGH T50HZ
                SETB    CR                  ;启动PCA 计时器
                SETB    EA

                JMP     $

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为11.0592MHz

#define   T50HZ         (11059200L / 12 / 2 / 50)

sfr     CCON    =   0xd8;
sbit    CF      =   CCON^7;
sbit    CR      =   CCON^6;
sbit    CCF3    =   CCON^3;
sbit    CCF2    =   CCON^2;
sbit    CCF1    =   CCON^1;
sbit    CCF0    =   CCON^0;
sfr     CMOD    =   0xd9;
sfr     CL      =   0xe9;
sfr     CH      =   0xf9;
sfr     CCAPM0  =   0xda;
sfr     CCAP0L  =   0xea;
sfr     CCAP0H  =   0xfa;
sfr     PCA_PWM0=   0xf2;
sfr     CCAPM1  =   0xdb;
sfr     CCAP1L  =   0xeb;
sfr     CCAP1H  =   0xfb;
```

```
sfr     PCA_PWM1    =   0xf3;
sfr     CCAPM2      =   0xdc;
sfr     CCAP2L      =   0xec;
sfr     CCAP2H      =   0xfc;
sfr     PCA_PWM2    =   0xf4;
sfr     CCAPM3      =   0xdd;
sfr     CCAP3L      =   0xed;
sfr     CCAP3H      =   0xfd;
sfr     PCA_PWM3    =   0xf5;

sbit    P10         =   P1^0;

unsigned int    value;

void PCA_Isr() interrupt 7
{
    CCF0 = 0;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T50HZ;

    P10 = !P10;                 //测试端口
}

void main()
{
    CCON = 0x00;
    CMOD = 0x00;                //PCA 时钟为系统时钟/12
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x49;              //PCA 模块 0 为 16 位定时器模式
    value = T50HZ;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T50HZ;
    CR = 1;                     //启动 PCA 计时器
    EA = 1;

    while (1);
}
```

17.4.4 PCA输出高速脉冲

汇编代码

```
;测试工作频率为 11.0592MHz

CCON        DATA    0D8H
CF          BIT     CCON.7
CR          BIT     CCON.6
CCF3        BIT     CCON.3
CCF2        BIT     CCON.2
CCF1        BIT     CCON.1
CCF0        BIT     CCON.0
CMOD        DATA    0D9H
CL          DATA    0E9H
CH          DATA    0F9H
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
```

```
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H
CCAPM3      DATA    0DDH
CCAP3L      DATA    0EDH
CCAP3H      DATA    0FDH
PCA_PWM3    DATA    0F5H

T38K4HZ     EQU     90H             ;11059200/2/38400

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            PUSH    ACC
            PUSH    PSW
            CLR     CCF0
            MOV     A,CCAP0L
            ADD     A,#LOW T38K4HZ
            MOV     CCAP0L,A
            MOV     A,CCAP0H
            ADDC    A,#HIGH T38K4HZ
            MOV     CCAP0H,A
            POP     PSW
            POP     ACC
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     CCON,#00H
            MOV     CMOD,#08H       ;PCA 时钟为系统时钟
            MOV     CL,#00H
            MOV     CH,#0H
            MOV     CCAPM0,#4DH     ;PCA 模块 0 为 16 位定时器模式并使能脉冲输出
            MOV     CCAP0L,#LOW T38K4HZ
            MOV     CCAP0H,#HIGH T38K4HZ
            SETB    CR              ;启动 PCA 计时器
            SETB    EA

            JMP     $

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"
```

```
//测试工作频率为11.0592MHz

#define   T38K4HZ         (11059200L / 2 / 38400)

sfr       CCON      =     0xd8;
sbit      CF        =     CCON^7;
sbit      CR        =     CCON^6;
sbit      CCF3      =     CCON^3;
sbit      CCF2      =     CCON^2;
sbit      CCF1      =     CCON^1;
sbit      CCF0      =     CCON^0;
sfr       CMOD      =     0xd9;
sfr       CL        =     0xe9;
sfr       CH        =     0xf9;
sfr       CCAPM0    =     0xda;
sfr       CCAP0L    =     0xea;
sfr       CCAP0H    =     0xfa;
sfr       PCA_PWM0  =     0xf2;
sfr       CCAPM1    =     0xdb;
sfr       CCAP1L    =     0xeb;
sfr       CCAP1H    =     0xfb;
sfr       PCA_PWM1  =     0xf3;
sfr       CCAPM2    =     0xdc;
sfr       CCAP2L    =     0xec;
sfr       CCAP2H    =     0xfc;
sfr       PCA_PWM2  =     0xf4;
sfr       CCAPM3    =     0xdd;
sfr       CCAP3L    =     0xed;
sfr       CCAP3H    =     0xfd;
sfr       PCA_PWM3  =     0xf5;

unsigned int      value;

void PCA_Isr() interrupt 7
{
      CCF0 = 0;
      CCAP0L = value;
      CCAP0H = value >> 8;
      value += T38K4HZ;
}

void main()
{
      CCON = 0x00;
      CMOD = 0x08;                  //PCA 时钟为系统时钟
      CL = 0x00;
      CH = 0x00;
      CCAPM0 = 0x4d;                //PCA 模块 0 为 16 位定时器模式并使能脉冲输出
      value = T38K4HZ;
      CCAP0L = value;
      CCAP0H = value >> 8;
      value += T38K4HZ;
      CR = 1;                       //启动 PCA 计时器
      EA = 1;

      while (1);
}
```

STC MCU

17.4.5 PCA扩展外部中断

汇编代码

```
;测试工作频率为11.0592MHz

CCON        DATA    0D8H
CF          BIT     CCON.7
CR          BIT     CCON.6
CCF3        BIT     CCON.3
CCF2        BIT     CCON.2
CCF1        BIT     CCON.1
CCF0        BIT     CCON.0
CMOD        DATA    0D9H
CL          DATA    0E9H
CH          DATA    0F9H
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H
CCAPM3      DATA    0DDH
CCAP3L      DATA    0EDH
CCAP3H      DATA    0FDH
PCA_PWM3    DATA    0F5H

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            CLR     CCF0
            CPL     P1.0
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     CCON,#00H
            MOV     CMOD,#08H       ;PCA 时钟为系统时钟
            MOV     CL,#00H
            MOV     CH,#0H
            MOV     CCAPM0,#11H     ;扩展外部端口CCP0 为下降沿中断口
;           MOV     CCAPM0,#21H     ;扩展外部端口CCP0 为上升沿中断口
;           MOV     CCAPM0,#31H     ;扩展外部端口CCP0 为边沿中断口
            MOV     CCAP0L,#0
            MOV     CCAP0H,#0
            SETB    CR              ;启动 PCA 计时器
            SETB    EA
```

```
        JMP     $

        END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为11.0592MHz

sfr     CCON    =   0xd8;
sbit    CF      =   CCON^7;
sbit    CR      =   CCON^6;
sbit    CCF3    =   CCON^3;
sbit    CCF2    =   CCON^2;
sbit    CCF1    =   CCON^1;
sbit    CCF0    =   CCON^0;
sfr     CMOD    =   0xd9;
sfr     CL      =   0xe9;
sfr     CH      =   0xf9;
sfr     CCAPM0  =   0xda;
sfr     CCAP0L  =   0xea;
sfr     CCAP0H  =   0xfa;
sfr     PCA_PWM0=   0xf2;
sfr     CCAPM1  =   0xdb;
sfr     CCAP1L  =   0xeb;
sfr     CCAP1H  =   0xfb;
sfr     PCA_PWM1=   0xf3;
sfr     CCAPM2  =   0xdc;
sfr     CCAP2L  =   0xec;
sfr     CCAP2H  =   0xfc;
sfr     PCA_PWM2=   0xf4;
sfr     CCAPM3  =   0xdd;
sfr     CCAP3L  =   0xed;
sfr     CCAP3H  =   0xfd;
sfr     PCA_PWM3=   0xf5;

sbit    P10     =   P1^0;

void PCA_Isr() interrupt 7
{
    CCF0 = 0;
    P10 = !P10;
}

void main()
{
    CCON = 0x00;
    CMOD = 0x08;                //PCA 时钟为系统时钟
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x11;              //扩展外部端口CCP0为下降沿中断口
//  CCAPM0 = 0x21;              //扩展外部端口CCP0为上升沿中断口
//  CCAPM0 = 0x31;              //扩展外部端口CCP0为边沿中断口
    CCAP0L = 0;
    CCAP0H = 0;
    CR = 1;                     //启动PCA 计时器
    EA = 1;
```

STC MCU

```
    while (1);
}
```

STC MCU

18 精度可达 15 位的增强型PWM

特别注意：增强型 PWM 的输出脚可设置为准双向口模式或者推挽输出模式，但发送脚一定不能设置开漏模式，否则没波形输出

STC8A 系列单片机集成了 1 组（各自独立 8 路）增强型的 PWM 波形发生器。PWM 的时钟源可以选择。PWM 波形发生器内部有一个 15 位的 PWM 计数器供 8 路 PWM 使用，用户可以设置每路 PWM 的初始电平。另外，PWM 波形发生器为每路 PWM 又设计了两个用于控制波形翻转的计数器 T1/T2，可以非常灵活的控制每路 PWM 的高低电平宽度，从而达到对 PWM 的占空比以及 PWM 的输出延迟进行控制的目的。由于每组的 8 路 PWM 是各自独立的，且每路 PWM 的初始状态可以进行设定，所以用户可以将其中的任意两路配合起来使用，即可实现互补对称输出以及死区控制等特殊应用。（注：增强型 PWM 只有输出功能，如果需要测量脉冲宽度，请使用本系列的 PCA/CCP/PWM 功能）增强型的 PWM 波形发生器还设计了对外部异常事件（包括外部 P3.5 端口电平异常、比较器比较结果异常）进行监控的功能，可用于紧急关闭 PWM 输出。PWM 波形发生器还可与 ADC 相关联，设置 PWM 周期的任一时间点触发 ADC 转换事件。

STC 三种硬件 PWM 比较：

兼容传统 8051 的 PCA/CCP/PWM：可输出 PWM 波形、捕获外部输入信号以及输出高速脉冲。可对外输出 6 位/7 位/8 位/10 位的 PWM 波形，6 位 PWM 波形的频率为 PCA 模块时钟源频率/64；7 位 PWM 波形的频率为 PCA 模块时钟源频率/128；8 位 PWM 波形的频率为 PCA 模块时钟源频率/256；10 位 PWM 波形的频率为 PCA 模块时钟源频率/1024。捕获外部输入信号，可捕获上升沿、下降沿或者同时捕获上升沿和下降沿。

STC8A/8G 系列的 15 位增强型 PWM：只能对外输出 PWM 波形，无输入捕获功能。对外输出 PWM 的频率以及占空比均可任意设置。通过软件干预，可实现多路互补/对称/带死区的 PWM 波形。有外部异常检测功能以及实时触发 ADC 转换功能。

STC8H 系列的 16 位高级 PWM 定时器：是目前 STC 功能最强的 PWM，可对外输出任意频率以及任意占空比的 PWM 波形。无需软件干预即可输出互补/对称/带死区的 PWM 波形。能捕获外部输入信号，可捕获上升沿、下降沿或者同时捕获上升沿和下降沿，测量外部波形时，可同时测量波形的周期值和占空比值。有正交编码功能、外部异常检测功能以及实时触发 ADC 转换功能。

18.1 PWM相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWMCFG	增强型 PWM 配置寄存器	F1H	CBIF	ETADC	-	-	-	-	-	-	00xx,xxxx
PWMIF	增强型 PWM 中断标志寄存器	F6H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWMFDCR	PWM 异常检测控制寄存器	F7H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWMCR	PWM 控制寄存器	FEH	ENPWM	ECBI	-	-	-	-	-	-	00xx,xxxx

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWMCH	PWM 计数器高字节	FFF0H	-								x000,0000
PWMCL	PWM 计数器低字节	FFF1H									0000,0000
PWMCKS	PWM 时钟选择	FFF2H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000

TADCPH	触发 ADC 计数值高字节	FFF3H	-							x000,0000
TADCPL	触发 ADC 计数值低字节	FFF4H								0000,0000
PWM0T1H	PWM0T1 计数值高字节	FF00H	-							x000,0000
PWM0T1L	PWM0T1 计数值低节	FF01H								0000,0000
PWM0T2H	PWM0T2 数值高字节	FF02H	-							x000,0000
PWM0T2L	PWM0T2 数值低节	FF03H								0000,0000
PWM0CR	PWM0 控制寄存器	FF04H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI	00x0,0000
PWM0HLD	PWM0 电平保持控制寄存器	FF05H	-	-	-	-	-	-	HC0H	HC0L	xxxx,xx00
PWM1T1H	PWM1T1 计数值高字节	FF10H	-							x000,0000
PWM1T1L	PWM1T1 计数值低节	FF11H								0000,0000
PWM1T2H	PWM1T2 数值高字节	FF12H	-							x000,0000
PWM1T2L	PWM1T2 数值低节	FF13H								0000,0000
PWM1CR	PWM1 控制寄存器	FF14H	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI	00x0,0000
PWM1HLD	PWM1 电平保持控制寄存器	FF15H	-	-	-	-	-	-	HC1H	HC1L	xxxx,xx00
PWM2T1H	PWM2T1 计数值高字节	FF20H	-							x000,0000
PWM2T1L	PWM2T1 计数值低节	FF21H								0000,0000
PWM2T2H	PWM2T2 数值高字节	FF22H	-							x000,0000
PWM2T2L	PWM2T2 数值低节	FF23H								0000,0000
PWM2CR	PWM2 控制寄存器	FF24H	ENC2O	C2INI	-	C2_S[1:0]		EC2I	EC2T2SI	EC2T1SI	00x0,0000
PWM2HLD	PWM2 电平保持控制寄存器	FF25H	-	-	-	-	-	-	HC2H	HC2L	xxxx,xx00
PWM3T1H	PWM3T1 计数值高字节	FF30H	-							x000,0000
PWM3T1L	PWM3T1 计数值低节	FF31H								0000,0000
PWM3T2H	PWM3T2 数值高字节	FF32H	-							x000,0000
PWM3T2L	PWM3T2 数值低节	FF33H								0000,0000
PWM3CR	PWM3 控制寄存器	FF34H	ENC3O	C3INI	-	C3_S[1:0]		EC3I	EC3T2SI	EC3T1SI	00x0,0000
PWM3HLD	PWM3 电平保持控制寄存器	FF35H	-	-	-	-	-	-	HC3H	HC3L	xxxx,xx00
PWM4T1H	PWM4T1 计数值高字节	FF40H	-							x000,0000
PWM4T1L	PWM4T1 计数值低节	FF41H								0000,0000
PWM4T2H	PWM4T2 数值高字节	FF42H	-							x000,0000
PWM4T2L	PWM4T2 数值低节	FF43H								0000,0000
PWM4CR	PWM4 控制寄存器	FF44H	ENC4O	C4INI	-	C4_S[1:0]		EC4I	EC4T2SI	EC4T1SI	00x0,0000
PWM4HLD	PWM4 电平保持控制寄存器	FF45H	-	-	-	-	-	-	HC4H	HC4L	xxxx,xx00
PWM5T1H	PWM5T1 计数值高字节	FF50H	-							x000,0000
PWM5T1L	PWM5T1 计数值低节	FF51H								0000,0000
PWM5T2H	PWM5T2 数值高字节	FF52H	-							x000,0000
PWM5T2L	PWM5T2 数值低节	FF53H								0000,0000
PWM5CR	PWM5 控制寄存器	FF54H	ENC5O	C5INI	-	C5_S[1:0]		EC5I	EC5T2SI	EC5T1SI	00x0,0000
PWM5HLD	PWM5 电平保持控制寄存器	FF55H	-	-	-	-	-	-	HC5H	HC5L	xxxx,xx00
PWM6T1H	PWM6T1 计数值高字节	FF60H	-							x000,0000
PWM6T1L	PWM6T1 计数值低节	FF61H								0000,0000
PWM6T2H	PWM6T2 数值高字节	FF62H	-							x000,0000
PWM6T2L	PWM6T2 数值低节	FF63H								0000,0000
PWM6CR	PWM6 控制寄存器	FF64H	ENC6O	C6INI	-	C6_S[1:0]		EC6I	EC6T2SI	EC6T1SI	00x0,0000
PWM6HLD	PWM6 电平保持控制寄存器	FF65H	-	-	-	-	-	-	HC6H	HC6L	xxxx,xx00

PWM7T1H	PWM7T1 计数值高字节	FF70H	-								x000,0000
PWM7T1L	PWM7T1 计数值低节	FF71H									0000,0000
PWM7T2H	PWM7T2 数值高字节	FF72H	-								x000,0000
PWM7T2L	PWM7T2 数值低节	FF73H									0000,0000
PWM7CR	PWM7 控制寄存器	FF74H	ENC7O	C7INI	-	C7_S[1:0]		EC7I	EC7T2SI	EC7T1SI	00x0,0000
PWM7HLD	PWM7 电平保持控制寄存器	FF75H	-	-	-	-	-	-	HC7H	HC7L	xxxx,xx00

18.1.1 增强型PWM配置寄存器（PWMCFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCFG	F1H	CBIF	ETADC	-	-	-	-	-	-

CBIF：PWM 计数器归零中断标志位。

当 15 位的 PWM 计数器记满溢出归零时，硬件自动将此位置 1，并向 CPU 提出中断请求，此标志位需要软件清零。

ETADC：PWM 是否与 ADC 关联

0：PWM 与 ADC 不关联

1：PWM 与 ADC 相关联。允许在 PWM 周期中某个时间点触发 A/D 转换。使用 TADCPH 和 TADCPL 进行设置。

18.1.2 PWM中断标志寄存器（PWMIF）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMIF	F6H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF

CnIF：第 n 通道 PWM 的中断标志位。

当所设置的触发点发生中断事件时，硬件自动将此位置 1，并向 CPU 提出中断请求，此标志位需要软件清零。

18.1.3 PWM异常检测控制寄存器（PWMFDCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMFDCR	F7H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF

INVCMP：比较器器结果异常信号处理

0：比较器器结果由低变高为异常信号

1：比较器器结果由高变低为异常信号

INVIO：外部端口 P3.5 异常信号处理

0：外部端口 P3.5 信号由低变高为异常信号

1：外部端口 P3.5 信号由高变低为异常信号

ENFD：PWM 外部异常检测控制位

0：关闭 PWM 外部异常检测功能

1：使能 PWM 外部异常检测功能

FLTFLIO：发生 PWM 外部异常时对 PWM 输出口控制位

0：发生 WM 外部异常时，PWM 的输出口不作任何改变

1：发生 WM 外部异常时，PWM 的输出口立即被设置为高阻输入模式。（注：只有 ENCnO=1 所对应的端口才会被强制悬空）

EFDI：PWM 异常检测中断使能位

0：关闭 PWM 异常检测中断（FDIF 依然会被硬件置位）

1：使能 PWM 异常检测中断

FDCMP：比较器输出异常检测使能位

0：比较器与 PWM 无关

1：设定 PWM 异常检测源为比较器输出（异常类型有 INVCMP 设定）

FDIO：P3.5 口电平异常检测使能位

0：P3.5 口电平与 PWM 无关

1：设定 PWM 异常检测源为 P3.5 口（异常类型有 INVIO 设定）

FDIF：PWM 异常检测中断标志位

当发生 PWM 异常（比较器的输出由低变高或者 P3.5 的电平由低变高）时，硬件自动将此位置 1。当 EFDI==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

18.1.4 PWM控制寄存器（PWMCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCR	FEH	ENPWM	ECBI	-	-	-	-	-	-

ENPWM：使能增强型 PWM 波形发生器

0：关闭 PWM 波形发生器

1：使能 PWM 波形发生器，PWM 计数器开始计数

关于 ENPWM 控制位的重要说明:

- ENPWM 一旦被使能后，内部的 PWM 计数器会立即开始计数，并与 T1/T2 两个触发点的值进行比较。所以 ENPWM 必须在其他所有的 PWM 设置（包括 T1/T2 触发点的设置、初始电平的设置、PWM 异常检测的设置以及 PWM 中断设置）都完成后，最后才能使能 ENPWM 位。
- ENPWM 控制位既是整个 PWM 模块的的使能位，也是 PWM 计数器开始计数的控制位。在 PWM 计数器计数的过程中，ENPWM 控制位被关闭时，PWM 计数会立即停止，当再次使能 ENPWM 控制位时，PWM 的计数会从 0 开始重新计数，而不会记忆 PWM 停止计数前的计数值

ECBI：PWM 计数器归零中断使能位

0：关闭 PWM 计数器归零中断（CBIF 依然会被硬件置位）

1：使能 PWM 计数器归零中断

18.1.5 PWM计数器寄存器（PWMCH，PWMCL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCH	FFF0H	-							
PWMCL	FFF1H								

PWM 计数器位一个 15 位的寄存器，可设定 1~32767 之间的任意值作为 PWM 的周期。PWM 波形发生器内部的计数器从 0 开始计数，每个 PWM 时钟周期递增 1，当内部计数器的计数值达到[PWMCH，PWMCL]所设定的 PWM 周期时，PWM 波形发生器内部的计数器将会从 0 重新开始开始计数，硬件会自动将 PWM 归零中断中断标志位 CBIF 置 1，若 ECBI=1，程序将跳转到相应中断入口执行中断服务程序。

18.1.6 PWM时钟选择寄存器（PWMCKS），输出频率计算公式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCKS	FFF2H	-	-	-	SELT2	PWM_PS[3:0]			

SELT2：PWM 时钟源选择

0：PWM 时钟源为系统时钟经分频器分频之后的时钟

1：PWM 时钟源为定时器 2 的溢出脉冲

PWM_PS[3:0]：系统时钟预分频参数

SELT2	PWM_PS[3:0]	PWM 输入时钟源频率
1	xxxx	定时器 2 的溢出脉冲
0	0000	SYSclk/1
0	0001	SYSclk/2
0	0010	SYSclk/3
…	…	…
0	x	SYSclk/(x+1)
…	…	…
0	1111	SYSclk/16

STC MCU

PWM 输出频率计算公式

时钟源选择（SELT2）	PWM输出频率计算公式
SELT2＝0 （系统时钟）	$\text{PWM输出频率} = \frac{\text{系统工作频率SYSclk}}{(\text{PWM_PS} + 1) \times [\text{PWMCH, PWMCL}]}$
SELT2＝1 （定时器2的溢出脉冲）	$\text{PWM输出频率} = \frac{\text{定时器2的溢出脉冲频率}}{[\text{PWMCH, PWMCL}]}$

18.1.7 PWM触发ADC计数器寄存器（PWMnTADC）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TADCPH	FFF3H	-							
TADCPL	FFF4H								

在 ETADC=1 且 ADC_POWER=1 时，{TADCPH,TADCPL}组成一个 15 位的寄存器。在 PWM 的计数周期中，当 PWM 的内部计数值与{TADCPH,TADCPL}的值相等时，硬件自动触发 A/D 转换。

18.1.8 PWM电平输出设置计数值寄存器（PWMnT1，PWMnT2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0T1H	FF00H	-							
PWM0T1L	FF01H								
PWM0T2H	FF02H	-							
PWM0T2L	FF03H								
PWM1T1H	FF10H	-							
PWM1T1L	FF11H								
PWM1T2H	FF12H	-							
PWM1T2L	FF13H								
PWM2T1H	FF20H	-							
PWM2T1L	FF21H								
PWM2T2H	FF22H	-							
PWM2T2L	FF23H								
PWM3T1H	FF30H	-							
PWM3T1L	FF31H								
PWM3T2H	FF32H	-							
PWM3T2L	FF33H								
PWM4T1H	FF40H	-							
PWM4T1L	FF41H								
PWM4T2H	FF42H	-							
PWM4T2L	FF43H								
PWM5T1H	FF50H	-							
PWM5T1L	FF51H								

STC MCU

PWM5T2H	FF52H	-	
PWM5T2L	FF53H		
PWM6T1H	FF60H	-	
PWM6T1L	FF61H		
PWM6T2H	FF62H	-	
PWM6T2L	FF63H		
PWM7T1H	FF70H	-	
PWM7T1L	FF71H		
PWM7T2H	FF72H	-	
PWM7T2L	FF73H		

PWM 每个通道的{PWMnT1H, PWMnT1L}和{PWMnT2H, PWMnT2L}分别组合成两个 15 位的寄存器，用于控制各路 PWM 每个周期中输出 PWM 波形的两个触发点。在 PWM 的计数周期中，当 PWM 的内部计数值与所设置的 T1 的值{PWMnT1H, PWMnT1L}相等时，PWM 的输出波形输出低电平；当 PWM 的内部计数值与所设置的 T2 的值{PWMnT2H, PWMnT2L}相等时，PWM 的输出波形输出高电平。

注意：当{PWMniT1H, PWMniT1L}与{PWMniT2H, PWMniT2L}的值设置相等时，若 PWM 的内部计数值与所设置的 T1/T2 的值相等，则会固定输出低电平。

18.1.9 PWM通道控制寄存器（PWMnCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0CR	FF04H	ENC0O	C0INI	-	C0_S[1:0]	EC0I	EC0T2SI	EC0T1SI	ENT1I
PWM1CR	FF14H	ENC1O	C1INI	-	C1_S[1:0]	EC1I	EC1T2SI	EC1T1SI	ENT1I
PWM2CR	FF24H	ENC2O	C2INI	-	C2_S[1:0]	EC2I	EC2T2SI	EC2T1SI	ENT1I
PWM3CR	FF34H	ENC3O	C3INI	-	C3_S[1:0]	EC3I	EC3T2SI	EC3T1SI	ENT1I
PWM4CR	FF44H	ENC4O	C4INI	-	C4_S[1:0]	EC4I	EC4T2SI	EC4T1SI	ENT1I
PWM5CR	FF54H	ENC5O	C5INI	-	C5_S[1:0]	EC5I	EC5T2SI	EC5T1SI	ENT1I
PWM6CR	FF64H	ENC6O	C6INI	-	C6_S[1:0]	EC6I	EC6T2SI	EC6T1SI	ENT1I
PWM7CR	FF74H	ENC7O	C7INI	-	C7_S[1:0]	EC7I	EC7T2SI	EC7T1SI	ENT1I

ENCnO：PWM 输出使能位

0：相应 PWM 通道的端口为 GPIO

1：相应 PWM 通道的端口为 PWM 输出口，受 PWM 波形发生器控制

CnINI：设置 PWM 输出端口的初始电平

0：第 n 通道的 PWM 初始电平为低电平

1：第 n 通道的 PWM 初始电平为高电平

Cn_S[1:0]：PWM 输出功能脚切换选择，请参考功能脚切换章节。

ECnI：第 n 通道的 PWM 中断使能控制位

0：关闭第 n 通道的 PWM 中断

1：使能第 n 通道的 PWM 中断

ECnT2SI：第 n 通道的 PWM 在第 2 个触发点中断使能控制位

0：关闭第 n 通道的 PWM 在第 2 个触发点中断

1：使能第 n 通道的 PWM 在第 2 个触发点中断

ECnT1SI：第 n 通道的 PWM 在第 1 个触发点中断使能控制位

0：关闭第 n 通道的 PWM 在第 1 个触发点中断

1：使能第 n 通道的 PWM 在第 1 个触发点中断

18.1.10 PWM通道电平保持控制寄存器（PWMnHLD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0HLD	FF05H	-	-	-	-	-	-	HC0H	HC0L
PWM1HLD	FF15H	-	-	-	-	-	-	HC1H	HC1L
PWM2HLD	FF25H	-	-	-	-	-	-	HC2H	HC2L
PWM3HLD	FF35H	-	-	-	-	-	-	HC3H	HC3L
PWM4HLD	FF45H	-	-	-	-	-	-	HC4H	HC4L
PWM5HLD	FF55H	-	-	-	-	-	-	HC5H	HC5L
PWM6HLD	FF65H	-	-	-	-	-	-	HC6H	HC6L
PWM7HLD	FF75H	-	-	-	-	-	-	HC7H	HC7L

HCnH：第 n 通道 PWM 强制输出高电平控制位

0：第 n 通道 PWM 正常输出

1：第 n 通道 PWM 强制输出高电平

HCnL：第 n 通道 PWM 强制输出低电平控制位

0：第 n 通道 PWM 正常输出

1：第 n 通道 PWM 强制输出低电平

注意事项：

1. **8 通道增强型 PWM 的输出，切换到 P1.0~P1.7 或 P2.0~P2.7，IO 设置为准双向口或推挽输出，PWM 输出正常，均为推挽输出，并且再直接操作 IO 将不影响 PWM 波形。但是如果将 IO 设置为开漏输出（允许内部上拉电阻或外接上拉电阻），则如果对应的 IO 输出高电平，PWM 无输出（IO 为高阻），而对应的 IO 输出低电平，PWM 有推挽输出。**

18.2 范例程序

18.2.1 输出任意周期和任意占空比的波形

汇编代码

```
;测试工作频率为11.0592MHz

P_SW2       DATA    0BAH

PWMCFG      DATA    0F1H
PWMIF       DATA    0F6H
PWMFDCR     DATA    0F7H
PWMCR       DATA    0FEH
PWMCH       XDATA   0FFF0H
PWMCL       XDATA   0FFF1H
PWMCKS      XDATA   0FFF2H
TADCPH      XDATA   0FFF3H
TADCPL      XDATA   0FFF4H
PWM0T1H     XDATA   0FF00H
PWM0T1L     XDATA   0FF01H
PWM0T2H     XDATA   0FF02H
PWM0T2L     XDATA   0FF03H
PWM0CR      XDATA   0FF04H
PWM0HLD     XDATA   0FF05H
PWM1T1H     XDATA   0FF10H
PWM1T1L     XDATA   0FF11H
PWM1T2H     XDATA   0FF12H
PWM1T2L     XDATA   0FF13H
PWM1CR      XDATA   0FF14H
PWM1HLD     XDATA   0FF15H
PWM2T1H     XDATA   0FF20H
PWM2T1L     XDATA   0FF21H
PWM2T2H     XDATA   0FF22H
PWM2T2L     XDATA   0FF23H
PWM2CR      XDATA   0FF24H
PWM2HLD     XDATA   0FF25H
PWM3T1H     XDATA   0FF30H
PWM3T1L     XDATA   0FF31H
PWM3T2H     XDATA   0FF32H
PWM3T2L     XDATA   0FF33H
PWM3CR      XDATA   0FF34H
PWM3HLD     XDATA   0FF35H
PWM4T1H     XDATA   0FF40H
PWM4T1L     XDATA   0FF41H
PWM4T2H     XDATA   0FF42H
PWM4T2L     XDATA   0FF43H
PWM4CR      XDATA   0FF44H
PWM4HLD     XDATA   0FF45H
PWM5T1H     XDATA   0FF50H
PWM5T1L     XDATA   0FF51H
PWM5T2H     XDATA   0FF52H
PWM5T2L     XDATA   0FF53H
PWM5CR      XDATA   0FF54H
PWM5HLD     XDATA   0FF55H
PWM6T1H     XDATA   0FF60H
PWM6T1L     XDATA   0FF61H
```

STC MCU

```
PWM6T2H     XDATA   0FF62H
PWM6T2L     XDATA   0FF63H
PWM6CR      XDATA   0FF64H
PWM6HLD     XDATA   0FF65H
PWM7T1H     XDATA   0FF70H
PWM7T1L     XDATA   0FF71H
PWM7T2H     XDATA   0FF72H
PWM7T2L     XDATA   0FF73H
PWM7CR      XDATA   0FF74H
PWM7HLD     XDATA   0FF75H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     P_SW2,#80H
            CLR     A
            MOV     DPTR,#PWMCKS
            MOVX    @DPTR,A             ;PWM 时钟为系统时钟
            MOV     A,#10H
            MOV     DPTR,#PWMCH         ;设置 PWM 周期为 1000H 个 PWM 时钟
            MOVX    @DPTR,A
            MOV     A,#00H
            MOV     DPTR,#PWMCL
            MOVX    @DPTR,A
            MOV     A,#01H
            MOV     DPTR,#PWM0T1H       ;在计数值为 100H 地方输出低电平
            MOVX    @DPTR,A
            MOV     A,#00H
            MOV     DPTR,#PWM0T1L
            MOVX    @DPTR,A
            MOV     A,#05H
            MOV     DPTR,#PWM0T2H       ;在计数值为 500H 地方输出高电平
            MOVX    @DPTR,A
            MOV     A,#00H
            MOV     DPTR,#PWM0T2L
            MOVX    @DPTR,A
            MOV     A,#80H
            MOV     DPTR,#PWM0CR        ;使能 PWM0 输出
            MOVX    @DPTR,A
            MOV     P_SW2,#00H

            MOV     PWMCR,#080H         ;启动 PWM 模块

            JMP     $

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为 11.0592MHz

sfr     P_SW2     =   0xba;
sfr     PWMCFG    =   0xf1;
sfr     PWMIF     =   0xf6;
```

```
sfr     PWMFDCR   =   0xf7;
sfr     PWMCR     =   0xfe;

#define PWMC            (*(unsigned int volatile xdata *)0xfff0)
#define PWMCKS          (*(unsigned char volatile xdata *)0xfff2)
#define TADCP           (*(unsigned int volatile xdata *)0xfff3)
#define PWM0T1          (*(unsigned int volatile xdata *)0xff00)
#define PWM0T2          (*(unsigned int volatile xdata *)0xff02)
#define PWM0CR          (*(unsigned char volatile xdata *)0xff04)
#define PWM0HLD         (*(unsigned char volatile xdata *)0xff05)
#define PWM1T1          (*(unsigned int volatile xdata *)0xff10)
#define PWM1T2          (*(unsigned int volatile xdata *)0xff12)
#define PWM1CR          (*(unsigned char volatile xdata *)0xff14)
#define PWM1HLD         (*(unsigned char volatile xdata *)0xff15)
#define PWM2T1          (*(unsigned int volatile xdata *)0xff20)
#define PWM2T2          (*(unsigned int volatile xdata *)0xff22)
#define PWM2CR          (*(unsigned char volatile xdata *)0xff24)
#define PWM2HLD         (*(unsigned char volatile xdata *)0xff25)
#define PWM3T1          (*(unsigned int volatile xdata *)0xff30)
#define PWM3T2          (*(unsigned int volatile xdata *)0xff32)
#define PWM3CR          (*(unsigned char volatile xdata *)0xff34)
#define PWM3HLD         (*(unsigned char volatile xdata *)0xff35)
#define PWM4T1          (*(unsigned int volatile xdata *)0xff40)
#define PWM4T2          (*(unsigned int volatile xdata *)0xff42)
#define PWM4CR          (*(unsigned char volatile xdata *)0xff44)
#define PWM4HLD         (*(unsigned char volatile xdata *)0xff45)
#define PWM5T1          (*(unsigned int volatile xdata *)0xff50)
#define PWM5T2          (*(unsigned int volatile xdata *)0xff52)
#define PWM5CR          (*(unsigned char volatile xdata *)0xff54)
#define PWM5HLD         (*(unsigned char volatile xdata *)0xff55)
#define PWM6T1          (*(unsigned int volatile xdata *)0xff60)
#define PWM6T2          (*(unsigned int volatile xdata *)0xff62)
#define PWM6CR          (*(unsigned char volatile xdata *)0xff64)
#define PWM6HLD         (*(unsigned char volatile xdata *)0xff65)
#define PWM7T1          (*(unsigned int volatile xdata *)0xff70)
#define PWM7T2          (*(unsigned int volatile xdata *)0xff72)
#define PWM7CR          (*(unsigned char volatile xdata *)0xff74)
#define PWM7HLD         (*(unsigned char volatile xdata *)0xff75)

void main()
{
    P_SW2 = 0x80;
    PWMCKS = 0x00;              // PWM 时钟为系统时钟
    PWMC = 0x1000;              //设置 PWM 周期为 1000H 个 PWM 时钟
    PWM0T1= 0x0100;             //在计数值为 100H 地方输出低电平
    PWM0T2= 0x0500;             //在计数值为 500H 地方输出高电平
    PWM0CR= 0x80;               //使能 PWM0 输出
    P_SW2 = 0x00;

    PWMCR = 0x80;               //启动 PWM 模块

    while (1);
}
```

18.2.2 两路PWM实现互补对称带死区控制的波形

汇编代码

```
;测试工作频率为 11.0592MHz
```

```
P_SW2       DATA    0BAH

PWMCFG      DATA    0F1H
PWMIF       DATA    0F6H
PWMFDCR     DATA    0F7H
PWMCR       DATA    0FEH
PWMCH       XDATA   0FFF0H
PWMCL       XDATA   0FFF1H
PWMCKS      XDATA   0FFF2H
TADCPH      XDATA   0FFF3H
TADCPL      XDATA   0FFF4H
PWM0T1H     XDATA   0FF00H
PWM0T1L     XDATA   0FF01H
PWM0T2H     XDATA   0FF02H
PWM0T2L     XDATA   0FF03H
PWM0CR      XDATA   0FF04H
PWM0HLD     XDATA   0FF05H
PWM1T1H     XDATA   0FF10H
PWM1T1L     XDATA   0FF11H
PWM1T2H     XDATA   0FF12H
PWM1T2L     XDATA   0FF13H
PWM1CR      XDATA   0FF14H
PWM1HLD     XDATA   0FF15H
PWM2T1H     XDATA   0FF20H
PWM2T1L     XDATA   0FF21H
PWM2T2H     XDATA   0FF22H
PWM2T2L     XDATA   0FF23H
PWM2CR      XDATA   0FF24H
PWM2HLD     XDATA   0FF25H
PWM3T1H     XDATA   0FF30H
PWM3T1L     XDATA   0FF31H
PWM3T2H     XDATA   0FF32H
PWM3T2L     XDATA   0FF33H
PWM3CR      XDATA   0FF34H
PWM3HLD     XDATA   0FF35H
PWM4T1H     XDATA   0FF40H
PWM4T1L     XDATA   0FF41H
PWM4T2H     XDATA   0FF42H
PWM4T2L     XDATA   0FF43H
PWM4CR      XDATA   0FF44H
PWM4HLD     XDATA   0FF45H
PWM5T1H     XDATA   0FF50H
PWM5T1L     XDATA   0FF51H
PWM5T2H     XDATA   0FF52H
PWM5T2L     XDATA   0FF53H
PWM5CR      XDATA   0FF54H
PWM5HLD     XDATA   0FF55H
PWM6T1H     XDATA   0FF60H
PWM6T1L     XDATA   0FF61H
PWM6T2H     XDATA   0FF62H
PWM6T2L     XDATA   0FF63H
PWM6CR      XDATA   0FF64H
PWM6HLD     XDATA   0FF65H
PWM7T1H     XDATA   0FF70H
PWM7T1L     XDATA   0FF71H
PWM7T2H     XDATA   0FF72H
PWM7T2L     XDATA   0FF73H
PWM7CR      XDATA   0FF74H
```

STC MCU

```
PWM7HLD     XDATA    0FF75H

            ORG      0000H
            LJMP     MAIN

            ORG      0100H
MAIN:
            MOV      P_SW2,#80H
            CLR      A
            MOV      DPTR,#PWMCKS
            MOVX     @DPTR,A          ;PWM 时钟为系统时钟
            MOV      A,#08H
            MOV      DPTR,#PWMCH      ;设置 PWM 周期为 0800H 个 PWM 时钟
            MOVX     @DPTR,A
            MOV      A,#00H
            MOV      DPTR,#PWMCL
            MOVX     @DPTR,A
            MOV      A,#01H
            MOV      DPTR,#PWM0T1H    ;PWM0 在计数值为 0100H 地方输出低电平
            MOVX     @DPTR,A
            MOV      A,#00H
            MOV      DPTR,#PWM0T1L
            MOVX     @DPTR,A
            MOV      A,#07H
            MOV      DPTR,#PWM0T2H    ;PWM0 在计数值为 0700H 地方输出高电平
            MOVX     @DPTR,A
            MOV      A,#00H
            MOV      DPTR,#PWM0T2L
            MOVX     @DPTR,A
            MOV      A,#00H
            MOV      DPTR,#PWM1T2H    ;PWM1 在计数值为 0080H 地方输出高电平
            MOVX     @DPTR,A
            MOV      A,#80H
            MOV      DPTR,#PWM1T2L
            MOVX     @DPTR,A
            MOV      A,#07H
            MOV      DPTR,#PWM1T1H    ;PWM1 在计数值为 0780H 地方输出低电平
            MOVX     @DPTR,A
            MOV      A,#80H
            MOV      DPTR,#PWM1T1L
            MOVX     @DPTR,A
            MOV      A,#080H
            MOV      DPTR,#PWM0CR     ;使能 PWM0 输出
            MOVX     @DPTR,A
            MOV      A,#80H
            MOV      DPTR,#PWM1CR     ;使能 PWM1 输出
            MOVX     @DPTR,A
            MOV      P_SW2,#00H

            MOV      PWMCR,#080H      ;启动 PWM 模块

            JMP      $

            END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"
```

```
//测试工作频率为11.0592MHz

sfr     P_SW2       =   0xba;
sfr     PWMCFG      =   0xf1;
sfr     PWMIF       =   0xf6;
sfr     PWMFDCR     =   0xf7;
sfr     PWMCR       =   0xfe;

#define PWMC        (*(unsigned int volatile xdata *)0xfff0)
#define PWMCKS      (*(unsigned char volatile xdata *)0xfff2)
#define TADCP       (*(unsigned int volatile xdata *)0xfff3)
#define PWM0T1      (*(unsigned int volatile xdata *)0xff00)
#define PWM0T2      (*(unsigned int volatile xdata *)0xff02)
#define PWM0CR      (*(unsigned char volatile xdata *)0xff04)
#define PWM0HLD     (*(unsigned char volatile xdata *)0xff05)
#define PWM1T1      (*(unsigned int volatile xdata *)0xff10)
#define PWM1T2      (*(unsigned int volatile xdata *)0xff12)
#define PWM1CR      (*(unsigned char volatile xdata *)0xff14)
#define PWM1HLD     (*(unsigned char volatile xdata *)0xff15)
#define PWM2T1      (*(unsigned int volatile xdata *)0xff20)
#define PWM2T2      (*(unsigned int volatile xdata *)0xff22)
#define PWM2CR      (*(unsigned char volatile xdata *)0xff24)
#define PWM2HLD     (*(unsigned char volatile xdata *)0xff25)
#define PWM3T1      (*(unsigned int volatile xdata *)0xff30)
#define PWM3T2      (*(unsigned int volatile xdata *)0xff32)
#define PWM3CR      (*(unsigned char volatile xdata *)0xff34)
#define PWM3HLD     (*(unsigned char volatile xdata *)0xff35)
#define PWM4T1      (*(unsigned int volatile xdata *)0xff40)
#define PWM4T2      (*(unsigned int volatile xdata *)0xff42)
#define PWM4CR      (*(unsigned char volatile xdata *)0xff44)
#define PWM4HLD     (*(unsigned char volatile xdata *)0xff45)
#define PWM5T1      (*(unsigned int volatile xdata *)0xff50)
#define PWM5T2      (*(unsigned int volatile xdata *)0xff52)
#define PWM5CR      (*(unsigned char volatile xdata *)0xff54)
#define PWM5HLD     (*(unsigned char volatile xdata *)0xff55)
#define PWM6T1      (*(unsigned int volatile xdata *)0xff60)
#define PWM6T2      (*(unsigned int volatile xdata *)0xff62)
#define PWM6CR      (*(unsigned char volatile xdata *)0xff64)
#define PWM6HLD     (*(unsigned char volatile xdata *)0xff65)
#define PWM7T1      (*(unsigned int volatile xdata *)0xff70)
#define PWM7T2      (*(unsigned int volatile xdata *)0xff72)
#define PWM7CR      (*(unsigned char volatile xdata *)0xff74)
#define PWM7HLD     (*(unsigned char volatile xdata *)0xff75)

void main()
{
    P_SW2 = 0x80;
    PWMCKS = 0x00;              // PWM 时钟为系统时钟
    PWMC = 0x0800;              //设置 PWM 周期为 0800H 个 PWM 时钟
    PWM0T1= 0x0100;             //PWM0 在计数值为 100H 地方输出低电平
    PWM0T2= 0x0700;             //PWM0 在计数值为 700H 地方输出高电平
    PWM1T2= 0x0080;             //PWM1 在计数值为 0080H 地方输出高电平
    PWM1T1= 0x0780;             //PWM1 在计数值为 0780H 地方输出低电平
    PWM0CR= 0x80;               //使能 PWM0 输出
    PWM1CR= 0x80;               //使能 PWM1 输出
    P_SW2 = 0x00;

    PWMCR = 0x80;               //启动 PWM 模块
```

```
    while (1);
}
```

18.2.3 PWM实现渐变灯（呼吸灯）

汇编代码

```
;测试工作频率为11.0592MHz

CYCLE       EQU     1000H

P_SW2       DATA    0BAH

PWMCFG      DATA    0F1H
PWMIF       DATA    0F6H
PWMFDCR     DATA    0F7H
PWMCR       DATA    0FEH
PWMCH       XDATA   0FFF0H
PWMCL       XDATA   0FFF1H
PWMCKS      XDATA   0FFF2H
TADCPH      XDATA   0FFF3H
TADCPL      XDATA   0FFF4H
PWM0T1H     XDATA   0FF00H
PWM0T1L     XDATA   0FF01H
PWM0T2H     XDATA   0FF02H
PWM0T2L     XDATA   0FF03H
PWM0CR      XDATA   0FF04H
PWM0HLD     XDATA   0FF05H
PWM1T1H     XDATA   0FF10H
PWM1T1L     XDATA   0FF11H
PWM1T2H     XDATA   0FF12H
PWM1T2L     XDATA   0FF13H
PWM1CR      XDATA   0FF14H
PWM1HLD     XDATA   0FF15H
PWM2T1H     XDATA   0FF20H
PWM2T1L     XDATA   0FF21H
PWM2T2H     XDATA   0FF22H
PWM2T2L     XDATA   0FF23H
PWM2CR      XDATA   0FF24H
PWM2HLD     XDATA   0FF25H
PWM3T1H     XDATA   0FF30H
PWM3T1L     XDATA   0FF31H
PWM3T2H     XDATA   0FF32H
PWM3T2L     XDATA   0FF33H
PWM3CR      XDATA   0FF34H
PWM3HLD     XDATA   0FF35H
PWM4T1H     XDATA   0FF40H
PWM4T1L     XDATA   0FF41H
PWM4T2H     XDATA   0FF42H
PWM4T2L     XDATA   0FF43H
PWM4CR      XDATA   0FF44H
PWM4HLD     XDATA   0FF45H
PWM5T1H     XDATA   0FF50H
PWM5T1L     XDATA   0FF51H
PWM5T2H     XDATA   0FF52H
PWM5T2L     XDATA   0FF53H
PWM5CR      XDATA   0FF54H
PWM5HLD     XDATA   0FF55H
```

STC MCU

```
PWM6T1H     XDATA   0FF60H
PWM6T1L     XDATA   0FF61H
PWM6T2H     XDATA   0FF62H
PWM6T2L     XDATA   0FF63H
PWM6CR      XDATA   0FF64H
PWM6HLD     XDATA   0FF65H
PWM7T1H     XDATA   0FF70H
PWM7T1L     XDATA   0FF71H
PWM7T2H     XDATA   0FF72H
PWM7T2L     XDATA   0FF73H
PWM7CR      XDATA   0FF74H
PWM7HLD     XDATA   0FF75H

DIR         BIT     20H.0
VALL        DATA    21H
VALH        DATA    22H

            ORG     0000H
            LJMP    MAIN
            ORG     00B3H
            LJMP    PWMISR

            ORG     0100H
PWMISR:
            PUSH    ACC
            PUSH    PSW
            PUSH    DPL
            PUSH    DPH
            PUSH    P_SW2

            MOV     P_SW2,#80H
            MOV     A,PWMCFG
            JNB     ACC.7,ISREXIT
            ANL     PWMCFG,#NOT 80H         ;清中断标志
            JNB     DIR,PWMDN
PWMUP:
            MOV     A,VALL
            ADD     A,#1
            MOV     VALL,A
            MOV     A,VALH
            ADDC    A,#0
            MOV     VALH,A
            CJNE    A,#HIGH CYCLE,SETPWM
            MOV     A,VALL
            CJNE    A,#LOW CYCLE,SETPWM
            CLR     DIR
            JMP     SETPWM
PWMDN:
            MOV     A,VALL
            ADD     A,#0FFH
            MOV     VALL,A
            MOV     A,VALH
            ADDC    A,#0FFH
            MOV     VALH,A
            JNZ     SETPWM
            MOV     A,VALL
            CJNE    A,#1,SETPWM
            SETB    DIR
SETPWM:
```

STC MCU

```
                MOV     A,VALH
                MOV     DPTR,#PWM0T2H
                MOVX    @DPTR,A
                MOV     A,VALL
                MOV     DPTR,#PWM0T2L
                MOVX    @DPTR,A
ISREXIT:
                POP     P_SW2
                POP     DPH
                POP     DPL
                POP     PSW
                POP     ACC
                RETI

MAIN:
                MOV     SP,#3FH

                SETB    DIR
                MOV     VALH,#00H
                MOV     VALL,#01H

                MOV     P_SW2,#80H
                CLR     A
                MOV     DPTR,#PWMCKS
                MOVX    @DPTR,A             ;PWM 时钟为系统时钟
                MOV     A,#HIGH CYCLE
                MOV     DPTR,#PWMCH         ;设置 PWM 周期
                MOVX    @DPTR,A
                MOV     A,#LOW CYCLE
                MOV     DPTR,#PWMCL
                MOVX    @DPTR,A
                MOV     A,#00H
                MOV     DPTR,#PWM0T1H
                MOVX    @DPTR,A
                MOV     A,#00H
                MOV     DPTR,#PWM0T1L
                MOVX    @DPTR,A
                MOV     A,VALH
                MOV     DPTR,#PWM0T2H
                MOVX    @DPTR,A
                MOV     A,VALL
                MOV     DPTR,#PWM0T2L
                MOVX    @DPTR,A
                MOV     A,#80H
                MOV     DPTR,#PWM0CR        ;使能 PWM0 输出
                MOVX    @DPTR,A
                MOV     P_SW2,#00H

                MOV     PWMCR,#0C0H         ;启动 PWM 模块并使能 PWM 中断
                SETB    EA
                JMP     $

                END
```

C 语言代码

```
#include "reg51.h"
#include "intrins.h"
```

```
//测试工作频率为11.0592MHz

#define   CYCLE             0x1000

sfr       P_SW2       =     0xba;
sfr       PWMCFG      =     0xf1;
sfr       PWMIF       =     0xf6;
sfr       PWMFDCR     =     0xf7;
sfr       PWMCR       =     0xfe;

#define   PWMC              (*(unsigned int volatile xdata *)0xfff0)
#define   PWMCKS            (*(unsigned char volatile xdata *)0xfff2)
#define   TADCP             (*(unsigned int volatile xdata *)0xfff3)
#define   PWM0T1            (*(unsigned int volatile xdata *)0xff00)
#define   PWM0T2            (*(unsigned int volatile xdata *)0xff02)
#define   PWM0CR            (*(unsigned char volatile xdata *)0xff04)
#define   PWM0HLD           (*(unsigned char volatile xdata *)0xff05)
#define   PWM1T1            (*(unsigned int volatile xdata *)0xff10)
#define   PWM1T2            (*(unsigned int volatile xdata *)0xff12)
#define   PWM1CR            (*(unsigned char volatile xdata *)0xff14)
#define   PWM1HLD           (*(unsigned char volatile xdata *)0xff15)
#define   PWM2T1            (*(unsigned int volatile xdata *)0xff20)
#define   PWM2T2            (*(unsigned int volatile xdata *)0xff22)
#define   PWM2CR            (*(unsigned char volatile xdata *)0xff24)
#define   PWM2HLD           (*(unsigned char volatile xdata *)0xff25)
#define   PWM3T1            (*(unsigned int volatile xdata *)0xff30)
#define   PWM3T2            (*(unsigned int volatile xdata *)0xff32)
#define   PWM3CR            (*(unsigned char volatile xdata *)0xff34)
#define   PWM3HLD           (*(unsigned char volatile xdata *)0xff35)
#define   PWM4T1            (*(unsigned int volatile xdata *)0xff40)
#define   PWM4T2            (*(unsigned int volatile xdata *)0xff42)
#define   PWM4CR            (*(unsigned char volatile xdata *)0xff44)
#define   PWM4HLD           (*(unsigned char volatile xdata *)0xff45)
#define   PWM5T1            (*(unsigned int volatile xdata *)0xff50)
#define   PWM5T2            (*(unsigned int volatile xdata *)0xff52)
#define   PWM5CR            (*(unsigned char volatile xdata *)0xff54)
#define   PWM5HLD           (*(unsigned char volatile xdata *)0xff55)
#define   PWM6T1            (*(unsigned int volatile xdata *)0xff60)
#define   PWM6T2            (*(unsigned int volatile xdata *)0xff62)
#define   PWM6CR            (*(unsigned char volatile xdata *)0xff64)
#define   PWM6HLD           (*(unsigned char volatile xdata *)0xff65)
#define   PWM7T1            (*(unsigned int volatile xdata *)0xff70)
#define   PWM7T2            (*(unsigned int volatile xdata *)0xff72)
#define   PWM7CR            (*(unsigned char volatile xdata *)0xff74)
#define   PWM7HLD           (*(unsigned char volatile xdata *)0xff75)

void PWM_Isr() interrupt 22
{
    static bit dir = 1;
    static int val = 0;

    if (PWMCFG & 0x80)
    {
        PWMCFG &= ~0x80;            //清中断标志
        if (dir)
        {
            val++;
            if (val >= CYCLE) dir = 0;
        }
```

```
            else
            {
                val--;
                if (val <= 1) dir = 1;
            }
            _push_(P_SW2);
            P_SW2 |= 0x80;
            PWM0T2 = val;
            _pop_(P_SW2);
        }
}

void main()
{
    P_SW2 = 0x80;
    PWMCKS = 0x00;              // PWM 时钟为系统时钟
    PWMC = CYCLE;               //设置 PWM 周期为
    PWM0T1= 0x0000;
    PWM0T2= 0x0001;
    PWM0CR= 0x80;               //使能 PWM0 输出
    P_SW2 = 0x00;

    PWMCR = 0xc0;               //启动 PWM 模块
    EA = 1;

    while (1);
}
```

STC MCU

19 同步串行外设接口SPI

特别注意：当 SPI 工作在主机模式时，MISO 脚可设置为准双向口模式或者输入口模式，SCLK 和 MOSI 脚可设置为准双向口模式或者推挽输出模式，但主机模式的 SCLK 和 MOSI 脚一定不能设置开漏模式，否则没波形输出；当 SPI 工作在从机模式时，SCLK 和 MOSI 脚可设置为准双向口模式或者输入口模式，MISO 脚可设置为准双向口模式或者推挽输出模式，但从机模式的 MISO 脚一定不能设置开漏模式，否则没波形输出

STC8A/8F 系列单片机内部集成了一种高速串行通信接口——SPI 接口。SPI 是一种全双工的高速同步通信总线。STC8A/8F 系列集成的 SPI 接口提供了两种操作模式：主模式和从模式。

19.1 SPI相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPCTL	SPI 控制寄存器	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]		0000,0100
SPDAT	SPI 数据寄存器	CFH									0000,0000

19.1.1 SPI状态寄存器（SPSTAT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	SPIF	WCOL	-	-	-	-	-	-

SPIF：SPI 中断标志位。

当发送/接收完成 1 字节的数据后，硬件自动将此位置 1，并向 CPU 提出中断请求。当 SSIG 位被设置为 0 时，由于 SS 管脚电平的变化而使得设备的主/从模式发生改变时，此标志位也会被硬件自动置 1，以标志设备模式发生变化。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

WCOL：SPI 写冲突标志位。

当 SPI 在进行数据传输的过程中写 SPDAT 寄存器时，硬件将此位置 1。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

19.1.2 SPI控制寄存器（SPCTL），SPI速度控制

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPCTL	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]	

SSIG：SS 引脚功能控制位

0：SS 引脚确定器件是主机还是从机

1：忽略 SS 引脚功能，使用 MSTR 确定器件是主机还是从机

SPEN：SPI 使能控制位

0：关闭 SPI 功能

1：使能 SPI 功能

DORD：SPI 数据位发送/接收的顺序

0：先发送/接收数据的高位（MSB）

1：先发送/接收数据的低位（LSB）

MSTR：器件主/从模式选择位

设置主机模式：

若 SSIG＝0，则 SS 管脚必须为高电平且设置 MSTR 为 1

若 SSIG＝1，则只需要设置 MSTR 为 1（忽略 SS 管脚的电平）

设置从机模式：

若 SSIG＝0，则 SS 管脚必须为低电平（与 MSTR 位无关）

若 SSIG＝1，则只需要设置 MSTR 为 0（忽略 SS 管脚的电平）

CPOL：SPI 时钟极性控制

0：SCLK 空闲时为低电平，SCLK 的前时钟沿为上升沿，后时钟沿为下降沿

1：SCLK 空闲时为高电平，SCLK 的前时钟沿为下降沿，后时钟沿为上升沿

CPHA：SPI 时钟相位控制

0：数据 SS 管脚为低电平驱动第一位数据并在 SCLK 的后时钟沿改变数据，前时钟沿采样数据（必须 SSIG＝0）

1：数据在 SCLK 的前时钟沿驱动，后时钟沿采样

SPR[1:0]：SPI 时钟频率选择

SPR[1:0]	SCLK 频率
00	SYSclk/4
01	SYSclk/8
10	SYSclk/16
11	SYSclk/32

19.1.3 SPI数据寄存器（SPDAT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPDAT	CFH								

SPI 发送/接收数据缓冲器。

STC MCU

19.2 SPI通信方式

SPI 的通信方式通常有 3 种：单主单从（一个主机设备连接一个从机设备）、互为主从（两个设备连接，设备和互为主机和从机）、单主多从（一个主机设备连接多个从机设备）

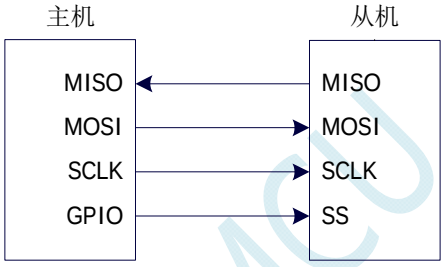
19.2.1 单主单从

两个设备相连，其中一个设备固定作为主机，另外一个固定作为从机。

主机设置：SSIG 设置为 1，MSTR 设置为 1，固定为主机模式。主机可以使用任意端口连接从机的 SS 管脚，拉低从机的 SS 脚即可使能从机

从机设置：SSIG 设置为 0，SS 管脚作为从机的片选信号。

单主单从连接配置图如下所示：



单主单从配置

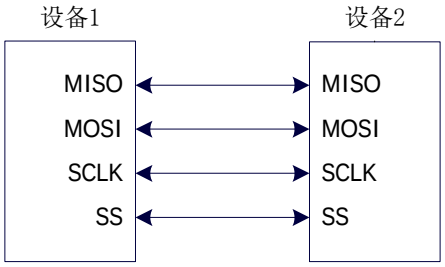
19.2.2 互为主从

两个设备相连，主机和从机不固定。

设置方法 1：两个设备初始化时都设置为 SSIG 设置为 0，MSTR 设置为 1，且将 SS 脚设置为双向口模式输出高电平。此时两个设备都是不忽略 SS 的主机模式。当其中一个设备需要启动传输时，可将自己的 SS 脚设置为输出模式并输出低电平，拉低对方的 SS 脚，这样另一个设备就被强行设置为从机模式了。

设置方法 2：两个设备初始化时都将自己设置成忽略 SS 的从机模式，即将 SSIG 设置为 1，MSTR 设置为 0。当其中一个设备需要启动传输时，先检测 SS 管脚的电平，如果时候高电平，就将自己设置成忽略 SS 的主模式，即可进行数据传输了。

互为主从连接配置图如下所示：



互为主从配置

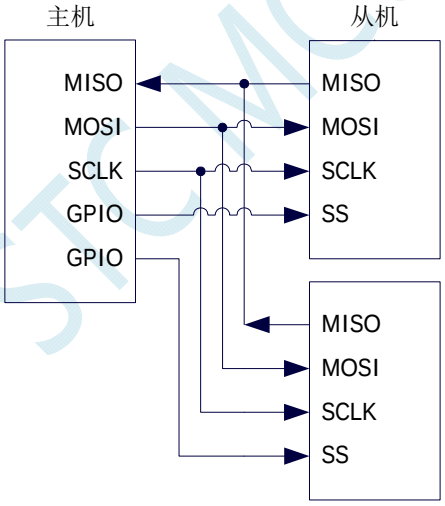
19.2.3 单主多从

多个设备相连，其中一个设备固定作为主机，其他设备固定作为从机。

主机设置：SSIG 设置为 1，MSTR 设置为 1，固定为主机模式。主机可以使用任意端口分别连接各个从机的 SS 管脚，拉低其中一个从机的 SS 脚即可使能相应的从机设备

从机设置：SSIG 设置为 0，SS 管脚作为从机的片选信号。

单主多从连接配置图如下所示：



单主多从配置

19.3 配置SPI

控制位			通信端口				说明
SPEN	SSIG	MSTR	SS	MISO	MOSI	SCLK	
0	x	x	x	输入	输入	输入	关闭 SPI 功能，SS/MOSI/MISO/SCLK 均为普通 I/O
1	0	0	0	输出	输入	输入	**从机模式**，且被选中
1	0	0	1	高阻	输入	输入	**从机模式**，但未被选中
1	0	1→0	0	输出	输入	输入	**从机模式**，不忽略 SS 且 MSTR 为 1 的主机模式，当 SS 管脚被拉低时，MSTR 将被硬件自动清零，工作模式将被被动设置为从机模式
1	0	1	1	输入	高阻	高阻	**主机模式**，空闲状态
					输出	输出	**主机模式**，激活状态
1	1	0	x	输出	输入	输入	**从机模式**
1	1	1	x	输入	输出	输出	**主机模式**

从机模式的注意事项：

当 CPHA＝0 时，SSIG 必须为 0（即不能忽略 SS 脚）。在每次串行字节开始还发送前 SS 脚必须拉低，并且在串行字节发送完后须重新设置为高电平。SS 管脚为低电平时不能对 SPDAT 寄存器执行写操作，否则将导致一个写冲突错误。CPHA=0 且 SSIG=1 时的操作未定义。

当 CPHA＝1 时，SSIG 可以置 1（即可以忽略脚）。如果 SSIG＝0，SS 脚可在连续传输之间保持低有效（即一直固定为低电平）。这种方式适用于固定单主单从的系统。

主机模式的注意事项：

在 SPI 中，传输总是由主机启动的。如果 SPI 使能（SPEN=1）并选择作为主机时，主机对 SPI 数据寄存器 SPDAT 的写操作将启动 SPI 时钟发生器和数据的传输。在数据写入 SPDAT 之后的半个到一个 SPI 位时间后，数据将出现在 MOSI 脚。写入主机 SPDAT 寄存器的数据从 MOSI 脚移出发送到从机的 MOSI 脚。同时从机 SPDAT 寄存器的数据从 MISO 脚移出发送到主机的 MISO 脚。

传输完一个字节后，SPI 时钟发生器停止，传输完成标志（SPIF）置位，如果 SPI 中断使能则会产生一个 SPI 中断。主机和从机 CPU 的两个移位寄存器可以看作是一个 16 位循环移位寄存器。当数据从主机移位传送到从机的同时，数据也以相反的方向移入。这意味着在一个移位周期中，主机和从机的数据相互交换。

通过 SS 改变模式

如果 SPEN=1，SSIG=0 且 MSTR=1，SPI 使能为主机模式，并将 SS 脚可配置为输入模式化或准双向口模式。这种情况下，另外一个主机可将该脚驱动为低电平，从而将该器件选择为 SPI 从机并向其发送数据。为了避免争夺总线，SPI 系统将该从机的 MSTR 清零，MOSI 和 SCLK 强制变为输入模式，而 MISO 则变为输出模式，同时 SPSTAT 的 SPIF 标志位置 1。

用户软件必须一直对 MSTR 位进行检测，如果该位被一个从机选择动作而被动清零，而用户想继续将 SPI 作为主机，则必须重新设置 MSTR 位，否则将一直处于从机模式。

写冲突

SPI 在发送时为单缓冲，在接收时为双缓冲。这样在前一次发送尚未完成之前，不能将新的数据写入移位寄存器。当发送过程中对数据寄存器 SPDAT 进行写操作时，WCOL 位将被置 1 以指示发生数据写冲突错误。在这种情况下，当前发送的数据继续发送，而新写入的数据将丢失。

当对主机或从机进行写冲突检测时，主机发生写冲突的情况是很罕见的，因为主机拥有数据传输的完全控制权。但从机有可能发生写冲突，因为当主机启动传输时，从机无法进行控制。

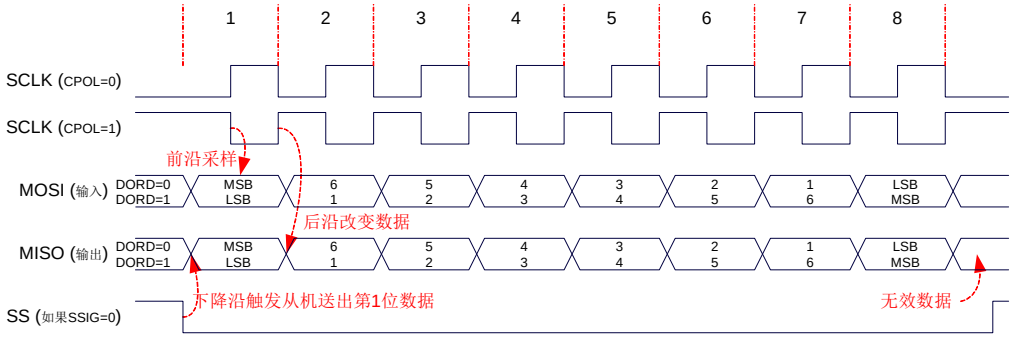
接收数据时，接收到的数据传送到一个并行读数据缓冲区，这样将释放移位寄存器以进行下一个数据的接收。但必须在下个字符完全移入之前从数据寄存器中读出接收到的数据，否则，前一个接收数据将丢失。

WCOL 可通过软件向其写入“1”清零。

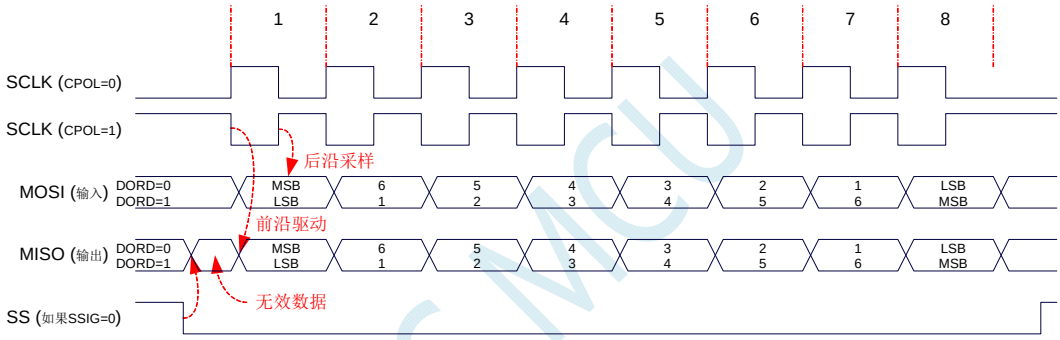
STC MCU

19.4 数据模式

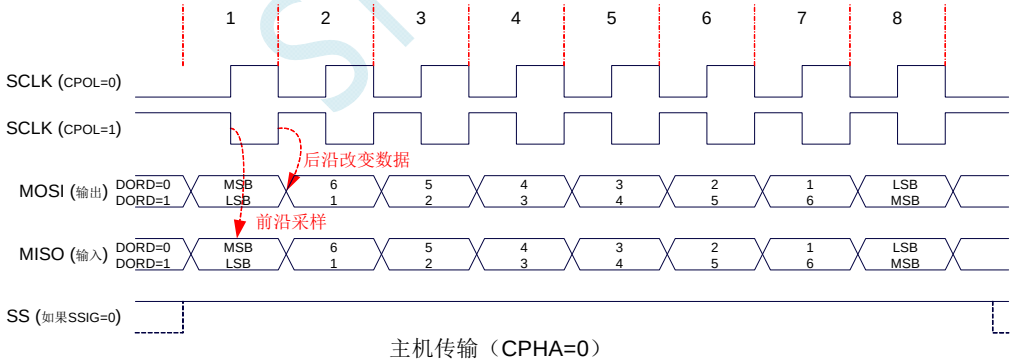
SPI 的时钟相位控制位 CPHA 可以让用户设定数据采样和改变时的时钟沿。时钟极性位 CPOL 可以让用户设定时钟极性。下面图例显示了不同时钟相位、极性设置下 SPI 通讯时序。



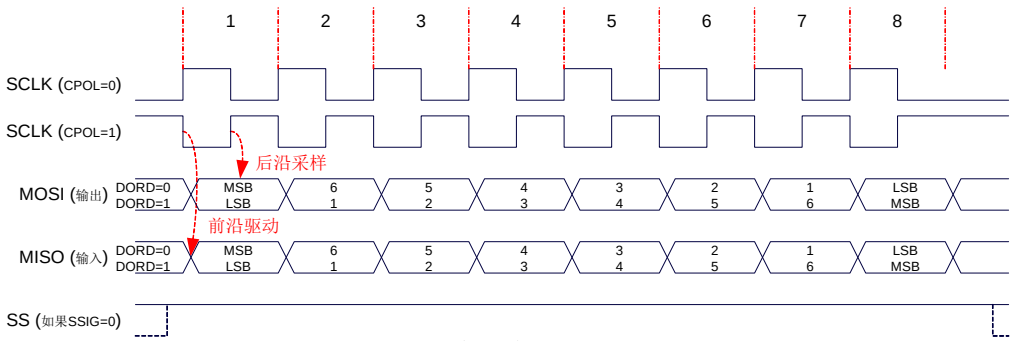
从机传输（CPHA=0）



从机传输（CPHA=1）



主机传输（CPHA=0）



主机传输（CPHA=1）

STC MCU

19.5 范例程序

19.5.1 SPI单主单从系统主机程序（中断方式）

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         SPSTAT      =   0xcd;
sfr         SPCTL       =   0xce;
sfr         SPDAT       =   0xcf;
sfr         IE2         =   0xaf;
#define     ESPI            0x02

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        SS          =   P1^0;
sbit        LED         =   P1^1;

bit         busy;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //清中断标志
    SS = 1;                         //拉高从机的SS 管脚
    busy = 0;
    LED = !LED;                     //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
```

```
    P5M1 = 0x00;

    LED = 1;
    SS = 1;
    busy = 0;

    SPCTL = 0x50;                           //使能 SPI 主机模式
    SPSTAT = 0xc0;                          //清中断标志
    IE2 = ESPI;                             //使能 SPI 中断
    EA = 1;

    while (1)
    {
        while (busy);
        busy = 1;
        SS = 0;                             //拉低从机 SS 管脚
        SPDAT = 0x5a;                       //发送测试数据
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

BUSY        BIT     20H.0
SS          BIT     P1.0
LED         BIT     P1.1

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            MOV     SPSTAT,#0C0H        ;清中断标志
            SETB    SS                  ;拉高从机的 SS 管脚
            CLR     BUSY
            CPL     LED
            RETI
```

```
MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        SETB    LED
        SETB    SS
        CLR     BUSY

        MOV     SPCTL,#50H          ;使能 SPI 主机模式
        MOV     SPSTAT,#0C0H        ;清中断标志
        MOV     IE2,#ESPI           ;使能 SPI 中断
        SETB    EA

LOOP:
        JB      BUSY,$
        SETB    BUSY
        CLR     SS                  ;拉低从机 SS 管脚
        MOV     SPDAT,#5AH          ;发送测试数据
        JMP     LOOP

        END
```

19.5.2 SPI单主单从系统从机程序（中断方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     SPSTAT      =   0xcd;
sfr     SPCTL       =   0xce;
sfr     SPDAT       =   0xcf;
sfr     IE2         =   0xaf;
#define ESPI            0x02

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
```

```
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    LED     =   P1^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //清中断标志
    SPDAT = SPDAT;                  //将接收到的数据回传给主机
    LED = !LED;                     //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x40;                   //使能 SPI 从机模式
    SPSTAT = 0xc0;                  //清中断标志
    IE2 = ESPI;                     //使能 SPI 中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

LED         BIT     P1.1

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
```

```
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     SPDAT,SPDAT         ;将接收到的数据回传给主机
            CPL     LED
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     SPCTL,#40H          ;使能 SPI 从机模式
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     IE2,#ESPI           ;使能 SPI 中断
            SETB    EA

            JMP     $

            END
```

19.5.3 SPI单主单从系统主机程序（查询方式）

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
```

```
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sfr     SPSTAT  =   0xcd;
sfr     SPCTL   =   0xce;
sfr     SPDAT   =   0xcf;
sfr     IE2     =   0xaf;
#define ESPI        0x02

sbit    SS      =   P1^0;
sbit    LED     =   P1^1;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    SS = 1;

    SPCTL = 0x50;                       //使能 SPI 主机模式
    SPSTAT = 0xc0;                      //清中断标志

    while (1)
    {
        SS = 0;                         //拉低从机 SS 管脚
        SPDAT = 0x5a;                   //发送测试数据
        while (!(SPSTAT & 0x80));       //查询完成标志
        SPSTAT = 0xc0;                  //清中断标志
        SS = 1;                         //拉高从机的SS 管脚
        LED = !LED;                     //测试端口
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT      DATA        0CDH
SPCTL       DATA        0CEH
SPDAT       DATA        0CFH
IE2         DATA        0AFH
ESPI        EQU         02H

SS          BIT         P1.0
LED         BIT         P1.1
```

```
P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            SETB    LED
            SETB    SS

            MOV     SPCTL,#50H          ;使能 SPI 主机模式
            MOV     SPSTAT,#0C0H        ;清中断标志

LOOP:
            CLR     SS                  ;拉低从机 SS 管脚
            MOV     SPDAT,#5AH          ;发送测试数据
            MOV     A,SPSTAT            ;查询完成标志
            JNB     ACC.7,$-2
            MOV     SPSTAT,#0C0H        ;清中断标志
            SETB    SS
            CPL     LED
            JMP     LOOP

            END
```

19.5.4 SPI单主单从系统从机程序（查询方式）

C 语言代码

```
//测试工作频率为 11.0592MHz
```

```
#include "reg51.h"
#include "intrins.h"

sfr         SPSTAT      =   0xcd;
sfr         SPCTL       =   0xce;
sfr         SPDAT       =   0xcf;
sfr         IE2         =   0xaf;
#define     ESPI            0x02

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        LED         =   P1^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                                  //清中断标志
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x40;                                   //使能 SPI 从机模式
    SPSTAT = 0xc0;                                  //清中断标志

    while (1)
    {
        while (!(SPSTAT & 0x80));                   //查询完成标志
        SPSTAT = 0xc0;                              //清中断标志
        SPDAT = SPDAT;                              //将接收到的数据回传给主机
        LED = !LED;                                 //测试端口
    }
}
```

汇编代码

```
;测试工作频率为11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

LED         BIT     P1.1

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     SPCTL,#40H          ;使能 SPI 从机模式
            MOV     SPSTAT,#0C0H        ;清中断标志

LOOP:
            MOV     A,SPSTAT            ;查询完成标志
            JNB     ACC.7,$-2
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     SPDAT,SPDAT         ;将接收到的数据回传给主机
            CPL     LED
            JMP     LOOP

            END
```

STC MCU

19.5.5 SPI互为主从系统程序（中断方式）

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         SPSTAT      =   0xcd;
sfr         SPCTL       =   0xce;
sfr         SPDAT       =   0xcf;
sfr         IE2         =   0xaf;
#define     ESPI            0x02

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        SS          =   P1^0;
sbit        LED         =   P1^1;
sbit        KEY         =   P0^0;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //清中断标志
    if (SPCTL & 0x10)
    {                               //主机模式
        SS = 1;                     //拉高从机的SS 管脚
        SPCTL = 0x40;               //重新设置为从机待机
    }
    else
    {                               //从机模式
        SPDAT = SPDAT;              //将接收到的数据回传给主机
    }
    LED = !LED;                     //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
```

STC MCU

```
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    KEY = 1;
    SS = 1;

    SPCTL = 0x40;                       //使能 SPI 从机模式进行待机
    SPSTAT = 0xc0;                      //清中断标志
    IE2 = ESPI;                         //使能 SPI 中断
    EA = 1;

    while (1)
    {
        if (!KEY)                       //等待按键触发
        {
            SPCTL = 0x50;               //使能 SPI 主机模式
            SS = 0;                     //拉低从机 SS 管脚
            SPDAT = 0x5a;               //发送测试数据
            while (!KEY);               //等待按键释放
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

SS          BIT     P1.0
LED         BIT     P1.1
KEY         BIT     P0.0

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
```

```
            PUSH    ACC
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     A,SPCTL
            JB      ACC.4,MASTER
SLAVE:
            MOV     SPDAT,SPDAT         ;将接收到的数据回传给主机
            JMP     ISREXIT
MASTER:
            SETB    SS                  ;拉高从机的 SS 管脚
            MOV     SPCTL,#40H          ;重新设置为从机待机
ISREXIT:
            CPL     LED
            POP     ACC
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            SETB    SS
            SETB    LED
            SETB    KEY

            MOV     SPCTL,#40H          ;使能 SPI 从机模式进行待机
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     IE2,#ESPI           ;使能 SPI 中断
            SETB    EA

LOOP:
            JB      KEY,LOOP            ;等待按键触发
            MOV     SPCTL,#50H          ;使能 SPI 主机模式
            CLR     SS                  ;拉低从机 SS 管脚
            MOV     SPDAT,#5AH          ;发送测试数据
            JNB     KEY,$               ;等待按键释放
            JMP     LOOP

            END
```

19.5.6 SPI互为主从系统程序（查询方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
```

```
#include "intrins.h"

sfr     SPSTAT      =   0xcd;
sfr     SPCTL       =   0xce;
sfr     SPDAT       =   0xcf;
sfr     IE2         =   0xaf;
#define ESPI            0x02

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sbit    SS          =   P1^0;
sbit    LED         =   P1^1;
sbit    KEY         =   P0^0;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    KEY = 1;
    SS = 1;

    SPCTL = 0x40;                   //使能 SPI 从机模式进行待机
    SPSTAT = 0xc0;                  //清中断标志

    while (1)
    {
        if (!KEY)                   //等待按键触发
        {
            SPCTL = 0x50;           //使能 SPI 主机模式
            SS = 0;                 //拉低从机 SS 管脚
            SPDAT = 0x5a;           //发送测试数据
            while (!KEY);           //等待按键释放
        }
        if (SPSTAT & 0x80)
        {
            SPSTAT = 0xc0;          //清中断标志
```

STC MCU

```
            if (SPCTL & 0x10)
            {                                   //主机模式
                SS = 1;                         //拉高从机的 SS 管脚
                SPCTL = 0x40;                   //重新设置为从机待机
            }
            else
            {                                   //从机模式
                SPDAT = SPDAT;                  //将接收到的数据回传给主机
            }
            LED = !LED;                         //测试端口
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

SS          BIT     P1.0
LED         BIT     P1.1
KEY         BIT     P0.0

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H
```

STC MCU

```
        SETB    SS
        SETB    LED
        SETB    KEY

        MOV     SPCTL,#40H          ;使能 SPI 从机模式进行待机
        MOV     SPSTAT,#0C0H        ;清中断标志

LOOP:
        JB      KEY,SKIP            ;等待按键触发
        MOV     SPCTL,#50H          ;使能 SPI 主机模式
        CLR     SS                  ;拉低从机 SS 管脚
        MOV     SPDAT,#5AH          ;发送测试数据
        JNB     KEY,$               ;等待按键释放
SKIP:
        MOV     A,SPSTAT
        JNB     ACC.7,LOOP
        MOV     SPSTAT,#0C0H        ;清中断标志
        MOV     A,SPCTL
        JB      ACC.4,MASTER
SLAVE:
        MOV     SPDAT,SPDAT         ;将接收到的数据回传给主机
        CPL     LED
        JMP     LOOP
MASTER:
        SETB    SS                  ;拉高从机的 SS 管脚
        MOV     SPCTL,#40H          ;重新设置为从机待机
        CPL     LED
        JMP     LOOP

        END
```

STC MCU

20 I²C总线

STC8A/8F 系列的单片机内部集成了一个 I²C 串行总线控制器。I²C 是一种高速同步通讯总线，通讯使用 SCL（时钟线）和 SDA（数据线）两线进行同步通讯。对于 SCL 和 SDA 的端口分配，STC8A/8F 系列的单片机提供了切换模式，可将 SCL 和 SDA 切换到不同的 I/O 口上，以方便用户将一组 I²C 总线当作多组进行分时复用。

与标准 I²C 协议相比较，忽略了如下两种机制：

- 发送起始信号（START）后不进行仲裁
- 时钟信号（SCL）停留在低电平时不进行超时检测

STC8A/8F 系列的 I²C 总线提供了两种操作模式：主机模式（SCL 为输出口，发送同步时钟信号）和从机模式（SCL 为输入口，接收同步时钟信号）

20.1 I²C相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
I2CCFG	I²C 配置寄存器	FE80H	ENI2C	MSSL	MSSPEED[5:0]						0000,0000
I2CMSCR	I²C 主机控制寄存器	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I²C 主机状态寄存器	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I²C 从机控制寄存器	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I²C 从机状态寄存器	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I²C 从机地址寄存器	FE85H	I2CSLADR[7:1]							MA	0000,0000
I2CTXD	I²C 数据发送寄存器	FE86H									0000,0000
I2CRXD	I²C 数据接收寄存器	FE87H									0000,0000
I2CMSAUX	I²C 主机辅助控制寄存器	FE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0

20.2 I²C主机模式

20.2.1 I2C配置寄存器（I2CCFG），总线速度控制

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CCFG	FE80H	ENI2C	MSSL	MSSPEED[5:0]					

ENI2C：I²C 功能使能控制位

0：禁止 I²C 功能

1：允许 I²C 功能

MSSL：I²C 工作模式选择位

0：从机模式

1：主机模式

MSSPEED[5:0]：I²C 总线速度（等待时钟数）控制，**I2C 总线速度＝F_{OSC} / 2 / (MSSPEED * 2 + 4)**

MSSPEED[5:0]	对应的时钟数
0	4
1	6
2	8
…	…
x	2x+4
…	…
62	128
63	130

只有当 I²C 模块工作在主机模式时，MSSPEED 参数设置的等待参数才有效。此等待参数主要用于主机模式的以下几个信号：

T_{SSTA}：起始信号的建立时间（Setup Time of START）

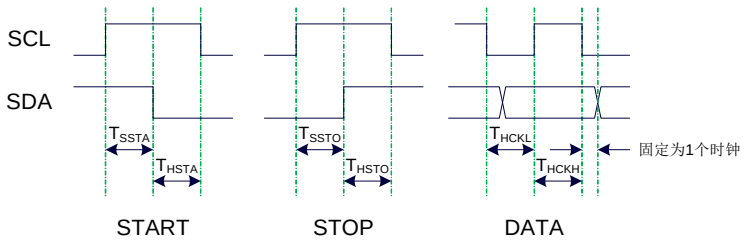
T_{HSTA}：起始信号的保持时间（Hold Time of START）

T_{SSTO}：停止信号的建立时间（Setup Time of STOP）

T_{HSTO}：停止信号的保持时间（Hold Time of STOP）

T_{HCKL}：时钟信号的低电平保持时间（Hold Time of SCL Low）

T_{HCKH}：时钟信号的高电平保持时间（Hold Time of SCL High）



例 1：当 MSSPEED＝10 时，T_{SSTA}＝T_{HSTA}＝T_{SSTO}＝T_{HSTO}＝T_{HCKL}＝24/FOSC

例 2：当 24MHz 的工作频率下需要 400K 的 I2C 总线速度时，

MSSPEED＝(24M / 400K / 2 - 4) / 2＝13

20.2.2 I2C主机控制寄存器（I2CMSCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	FE81H	EMSI	-	-	-	MSCMD[3:0]			

EMSI：主机模式中断使能控制位

0：关闭主机模式的中断

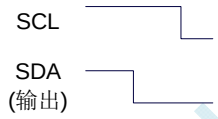
1：允许主机模式的中断

MSCMD[3:0]：主机命令

0000：待机，无动作。

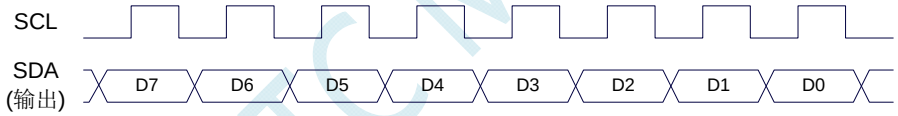
0001：起始命令。

发送 START 信号。如果当前 I²C 控制器处于空闲状态，即 MSBUSY（I2CMSST.7）为 0 时，写此命令会使控制器进入忙状态，硬件自动将 MSBUSY 状态位置 1，并开始发送 START 信号；**若当前 I²C 控制器处于忙状态，写此命令可触发发送 START 信号**。发送 START 信号的波形如下图所示：



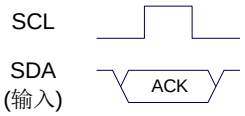
0010：发送数据命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 8 个时钟，并将 I2CTXD 寄存器里面数据按位送到 SDA 管脚上（先发送高位数据）。发送数据的波形如下图所示：



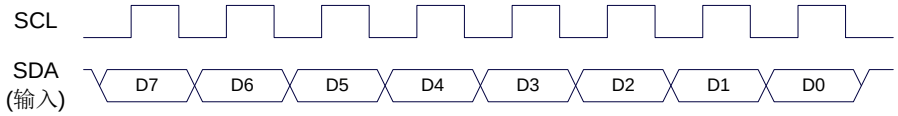
0011：接收 ACK 命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 1 个时钟，并将从 SDA 端口上读取的数据保存到 MSACKI（I2CMSST.1）。接收 ACK 的波形如下图所示：



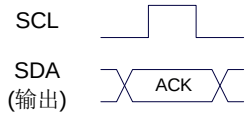
0100：接收数据命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 8 个时钟，并将从 SDA 端口上读取的数据依次左移到 I2CRXD 寄存器（先接收高位数据）。接收数据的波形如下图所示：



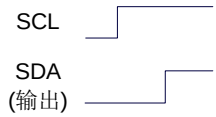
0101：发送 ACK 命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 1 个时钟，并将 MSACKO（I2CMSST.0）中的数据发送到 SDA 端口。发送 ACK 的波形如下图所示：



0110：停止命令。

发送 STOP 信号。写此命令后，I^2C 总线控制器开始发送 STOP 信号。信号发送完成后，硬件自动将 MSBUSY 状态位清零。STOP 信号的波形如下图所示：



0111：保留。

1000：保留。

1001：起始命令+发送数据命令+接收 ACK 命令。

此命令为命令 0001、命令 0010、命令 0011 三个命令的组合，下此命令后控制器会依次执行这三个命令。

1010：发送数据命令+接收 ACK 命令。

此命令为命令 0010、命令 0011 两个命令的组合，下此命令后控制器会依次执行这两个命令。

1011：接收数据命令+发送 ACK(0)命令。

此命令为命令 0100、命令 0101 两个命令的组合，下此命令后控制器会依次执行这两个命令。

注意：此命令所返回的应答信号固定为 ACK（0），不受 MSACKO 位的影响。

1100：接收数据命令+发送 NAK(1)命令。

此命令为命令 0100、命令 0101 两个命令的组合，下此命令后控制器会依次执行这两个命令。

注意：此命令所返回的应答信号固定为 NAK（1），不受 MSACKO 位的影响。

20.2.3 I2C主机辅助控制寄存器（I2CMSAUX）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSAUX	FE88H	-	-	-	-	-	-	-	WDTA

WDTA：主机模式时 I^2C 数据自动发送允许位

0：禁止自动发送

1：使能自动发送

若自动发送功能被使能，当 MCU 执行完成对 I2CTXD 数据寄存器的写操作后，I^2C 控制器会自动触发“1010”命令，即自动发送数据并接收 ACK 信号。

20.2.4 I2C主机状态寄存器（I2CMSST）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO

MSBUSY：主机模式时 I^2C 控制器状态位（只读位）

0：控制器处于空闲状态

1：控制器处于忙碌状态

当 I^2C 控制器处于主机模式时，在空闲状态下，发送完成 START 信号后，控制器便进入到忙碌状态，忙碌状态会一直维持到成功发送完成 STOP 信号，之后状态会再次恢复到空闲状态。

MSIF：主机模式的中断请求位（中断标志位）。当处于主机模式的 I^2C 控制器执行完成寄存器 I2CMSCR

中 MSCMD 命令后产生中断信号，硬件自动将此位 1，向 CPU 发请求中断，响应中断后 MSIF 位必须用软件清零。

MSACKI：主机模式时，发送“0011”命令到 I2CMSCR 的 MSCMD 位后所接收到的 ACK 数据。

MSACKO：主机模式时，准备将要发送出去的 ACK 信号。当发送“0101”命令到 I2CMSCR 的 MSCMD 位后，控制器会自动读取此位的数据当作 ACK 发送到 SDA。

STC MCU

20.3 I²C从机模式

20.3.1 I2C从机控制寄存器（I2CSLCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLCR	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

ESTAI：从机模式时接收到 START 信号中断允许位

0：禁止从机模式时接收到 START 信号时发生中断

1：使能从机模式时接收到 START 信号时发生中断

ERXI：从机模式时接收到 1 字节数据后中断允许位

0：禁止从机模式时接收到数据后发生中断

1：使能从机模式时接收到 1 字节数据后发生中断

ETXI：从机模式时发送完成 1 字节数据后中断允许位

0：禁止从机模式时发送完成数据后发生中断

1：使能从机模式时发送完成 1 字节数据后发生中断

ESTOI：从机模式时接收到 STOP 信号中断允许位

0：禁止从机模式时接收到 STOP 信号时发生中断

1：使能从机模式时接收到 STOP 信号时发生中断

SLRST：复位从机模式

20.3.2 I2C从机状态寄存器（I2CSLST）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLST	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	-	SLACKI	SLACKO

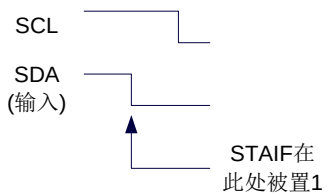
SLBUSY：从机模式时 I²C 控制器状态位（只读位）

0：控制器处于空闲状态

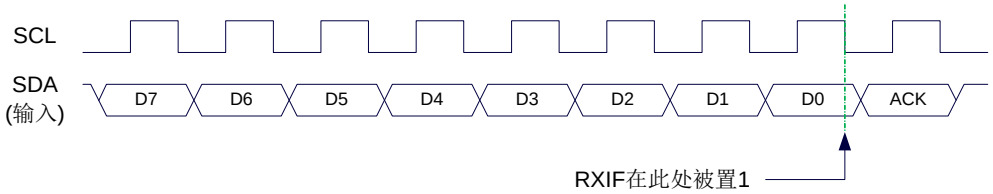
1：控制器处于忙碌状态

当 I²C 控制器处于从机模式时，在空闲状态下，接收到主机发送 START 信号后，控制器会继续检测之后的设备地址数据，若设备地址与当前 I2CSLADR 寄存器中所设置的从机地址相同时，控制器便进入到忙碌状态，忙碌状态会一直维持到成功接收到主机发送 STOP 信号，之后状态会再次恢复到空闲状态。

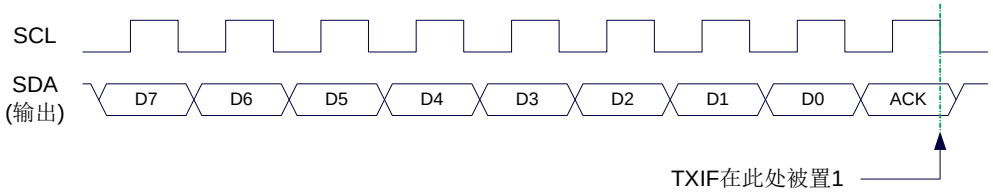
STAIF：从机模式时接收到 START 信号后的中断请求位。从机模式的 I²C 控制器接收到 START 信号后，硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 STAIF 位必须用软件清零。STAIF 被置 1 的时间点如下图所示：



RXIF：从机模式时接收到 1 字节的数据后的中断请求位。从机模式的 I²C 控制器接收到 1 字节的数据后，在第 8 个时钟的下降沿时硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 RXIF 位必须用软件清零。RXIF 被置 1 的时间点如下图所示：



TXIF：从机模式时发送完成 1 字节的数据后的中断请求位。从机模式的 I^2C 控制器发送完成 1 字节的数据并成功接收到 1 位 ACK 信号后，在第 9 个时钟的下降沿时硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 TXIF 位必须用软件清零。TXIF 被置 1 的时间点如下图所示：

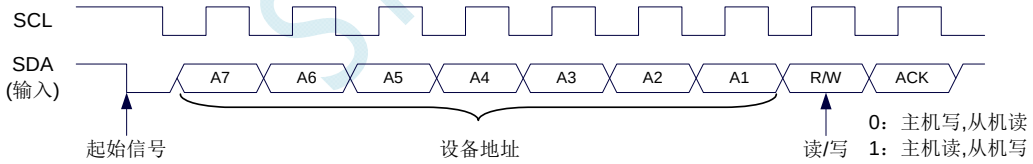


STOIF：从机模式时接收到 STOP 信号后的中断请求位。从机模式的 I^2C 控制器接收到 STOP 信号后，硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 STOIF 位必须用软件清零。STOIF 被置 1 的时间点如下图所示：



SLACKI：从机模式时，接收到的 ACK 数据。

SLACKO：从机模式时，准备将要发送出去的 ACK 信号。



20.3.3 I2C从机地址寄存器（I2CSLADR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLADR	FE85H	I2CSLADR[7:1]							MA

I2CSLADR[7:1]：从机设备地址

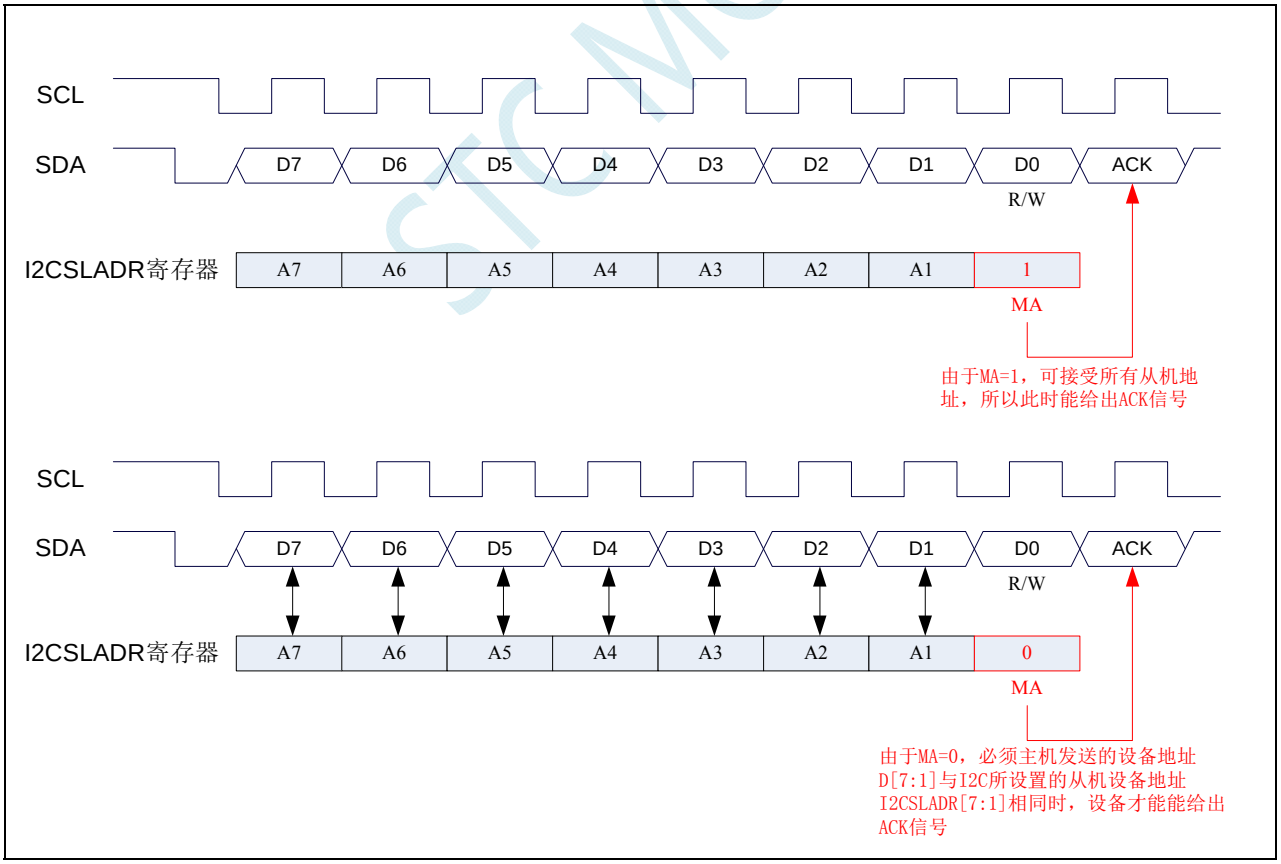
当 I^2C 控制器处于从机模式时，控制器在接收到 START 信号后，会继续检测接下来主机发送出的设备地址数据以及读/写信号。当主机发送出的设备地址与 I2CSLADR[7:1]中所设置的从机设备地址相同时，控制器才会向 CPU 发出中断求，请求 CPU 处理 I^2C 事件；否则若设备地址不同，I^2C 控制器继续监控，等待下一个起始信号，对下一个设备地址继续比较。

MA：从机设备地址比较控制

0：设备地址必须与 I2CSLADR[7:1]相同

1：忽略 I2CSLADR[7:1]中的设置，接受所有的设备地址

说明：I2C 总线协议规定 I2C 总线上最多可挂载 128 个 I2C 设备（理论值），不同的 I2C 设备用不同的 I2C 从机设备地址进行识别。I2C 主机发送完成起始信号后，发送的第一个数据（DATA0）的高 7 位即为从机设备地址（DATA0[7:1]为 I2C 设备地址），最低位为读写信号。当 I2C 设备从机地址寄存器 MA（I2CSLADR.0）为 1 时，表示 I2C 从机能够接受所有的设备地址，此时主机发送的任何设备地址，即 DATA0[7:1]为任何值，从机都能响应。当 I2C 设备从机地址寄存器 MA（I2CSLADR.0）为 0 时，主机发送的设备地址 DATA0[7:1]必须与从机的设备地址 I2CSLADR[7:1]相同时才能访问此从机设备



20.3.4 I2C数据寄存器（I2CTXD，I2CRXD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CTXD	FE86H								
I2CRXD	FE87H								

I2CTXD 是 I^2C 发送数据寄存器，存放将要发送的 I^2C 数据

I2CRXD 是 I^2C 接收数据寄存器，存放接收完成的 I^2C 数据

STC MCU

20.4 范例程序

20.4.1 I²C主机模式访问AT24C256（中断方式）

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P_SW2           =   0xba;

#define     I2CCFG              (*(unsigned char volatile xdata *)0xfe80)
#define     I2CMSCR             (*(unsigned char volatile xdata *)0xfe81)
#define     I2CMSST             (*(unsigned char volatile xdata *)0xfe82)
#define     I2CSLCR             (*(unsigned char volatile xdata *)0xfe83)
#define     I2CSLST             (*(unsigned char volatile xdata *)0xfe84)
#define     I2CSLADR            (*(unsigned char volatile xdata *)0xfe85)
#define     I2CTXD              (*(unsigned char volatile xdata *)0xfe86)
#define     I2CRXD              (*(unsigned char volatile xdata *)0xfe87)

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        SDA             =   P1^4;
sbit        SCL             =   P1^5;

bit         busy;

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;                       //清中断标志
        busy = 0;
    }
    _pop_(P_SW2);
}

void Start()
{
    busy = 1;
    I2CMSCR = 0x81;                             //发送 START 命令
```

```
    while (busy);
}

void SendData(char dat)
{
    I2CTXD = dat;                           //写数据到数据缓冲区
    busy = 1;
    I2CMSCR = 0x82;                         //发送 SEND 命令
    while (busy);
}

void RecvACK()
{
    busy = 1;
    I2CMSCR = 0x83;                         //发送读 ACK 命令
    while (busy);
}

char RecvData()
{
    busy = 1;
    I2CMSCR = 0x84;                         //发送 RECV 命令
    while (busy);
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                         //设置 ACK 信号
    busy = 1;
    I2CMSCR = 0x85;                         //发送 ACK 命令
    while (busy);
}

void SendNAK()
{
    I2CMSST = 0x01;                         //设置 NAK 信号
    busy = 1;
    I2CMSCR = 0x85;                         //发送 ACK 命令
    while (busy);
}

void Stop()
{
    busy = 1;
    I2CMSCR = 0x86;                         //发送 STOP 命令
    while (busy);
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
```

STC MCU

```
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0xe0;                          //使能 I2C 主机模式
    I2CMSST = 0x00;
    EA = 1;

    Start();    //发送起始命令
    SendData(0xa0);                         //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                         //发送存储地址高字节
    RecvACK();
    SendData(0x00);                         //发送存储地址低字节
    RecvACK();
    SendData(0x12);                         //写测试数据 1
    RecvACK();
    SendData(0x78);                         //写测试数据 2
    RecvACK();
    Stop();                                 //发送停止命令

    Delay();                                //等待设备写数据

    Start();                                //发送起始命令
    SendData(0xa0);                         //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                         //发送存储地址高字节
    RecvACK();
    SendData(0x00);                         //发送存储地址低字节
    RecvACK();
    Start();                                //发送起始命令
    SendData(0xa1);                         //发送设备地址+读命令
    RecvACK();
    P0 = RecvData();                        //读取数据 1
    SendACK();
    P2 = RecvData();                        //读取数据 2
    SendNAK();
    Stop();                                 //发送停止命令

    P_SW2 = 0x00;

    while (1);
}
```

汇编代码

```
;测试工作频率为11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

BUSY        BIT     20H.0

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00C3H
            LJMP    I2CISR

            ORG     0100H
I2CISR:
            PUSH    ACC
            PUSH    DPL
            PUSH    DPH

            MOV     DPTR,#I2CMSST       ;清中断标志
            MOVX    A,@DPTR
            ANL     A,#NOT 40H
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            CLR     BUSY                ;复位忙标志

            POP     DPH
            POP     DPL
            POP     ACC
            RETI

START:
            SETB    BUSY
```

STC MCU

```
                MOV     A,#10000001B            ;发送 START 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
SENDDATA:
                MOV     DPTR,#I2CTXD            ;写数据到数据缓冲区
                MOVX    @DPTR,A
                SETB    BUSY
                MOV     A,#10000010B            ;发送 SEND 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
RECVACK:
                SETB    BUSY
                MOV     A,#10000011B            ;发送读 ACK 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
RECVDATA:
                SETB    BUSY
                MOV     A,#10000100B            ;发送 RECV 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                CALL    WAIT
                MOV     DPTR,#I2CRXD            ;从数据缓冲区读取数据
                MOVX    A,@DPTR
                RET
SENDACK:
                MOV     A,#00000000B            ;设置 ACK 信号
                MOV     DPTR,#I2CMSST
                MOVX    @DPTR,A
                SETB    BUSY
                MOV     A,#10000101B            ;发送 ACK 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
SENDNAK:
                MOV     A,#00000001B            ;设置 NAK 信号
                MOV     DPTR,#I2CMSST
                MOVX    @DPTR,A
                SETB    BUSY
                MOV     A,#10000101B            ;发送 ACK 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
STOP:
                SETB    BUSY
                MOV     A,#10000110B            ;发送 STOP 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
WAIT:
                JB      BUSY,$                  ;等待命令发送完成
                RET

DELAY:
                MOV     R0,#0
                MOV     R1,#0
DELAY1:
```

```
        NOP
        NOP
        NOP
        NOP
        DJNZ    R1,DELAY1
        DJNZ    R0,DELAY1
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#80H

        MOV     A,#11100000B            ;设置 I2C 模块为主机模式
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        SETB    EA

        CALL    START                   ;发送起始命令
        MOV     A,#0A0H
        CALL    SENDDATA                ;发送设备地址+写命令
        CALL    RECVACK
        MOV     A,#000H                 ;发送存储地址高字节
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H                 ;发送存储地址低字节
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#12H                  ;写测试数据 1
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#78H                  ;写测试数据 2
        CALL    SENDDATA
        CALL    RECVACK
        CALL    STOP                    ;发送停止命令

        CALL    DELAY                   ;等待设备写数据

        CALL    START                   ;发送起始命令
        MOV     A,#0A0H                 ;发送设备地址+写命令
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H                 ;发送存储地址高字节
        CALL    SENDDATA
```

```
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址低字节
        CALL    SENDDATA
        CALL    RECVACK
        CALL    START               ;发送起始命令
        MOV     A,#0A1H             ;发送设备地址+读命令
        CALL    SENDDATA
        CALL    RECVACK
        CALL    RECVDATA            ;读取数据 1
        MOV     P0,A
        CALL    SENDACK
        CALL    RECVDATA            ;读取数据 2
        MOV     P2,A
        CALL    SENDNAK
        CALL    STOP                ;发送停止命令

        JMP     $

        END
```

20.4.2 I²C主机模式访问AT24C256（查询方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sbit    SDA         =   P1^4;
sbit    SCL         =   P1^5;
```

```
void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                         //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                           //写数据到数据缓冲区
    I2CMSCR = 0x02;                         //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                         //发送读 ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                         //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                         //设置 ACK 信号
    I2CMSCR = 0x05;                         //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                         //设置 NAK 信号
    I2CMSCR = 0x05;                         //发送 ACK 命令
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                         //发送 STOP 命令
    Wait();
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
```

```
            _nop_();
            _nop_();
            _nop_();
        }
}

void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        P_SW2 = 0x80;

        I2CCFG = 0xe0;                          //使能 I2C 主机模式
        I2CMSST = 0x00;

        Start();                                //发送起始命令
        SendData(0xa0);                         //发送设备地址+写命令
        RecvACK();
        SendData(0x00);                         //发送存储地址高字节
        RecvACK();
        SendData(0x00);                         //发送存储地址低字节
        RecvACK();
        SendData(0x12);                         //写测试数据 1
        RecvACK();
        SendData(0x78);                         //写测试数据 2
        RecvACK();
        Stop();                                 //发送停止命令

        Delay();                                //等待设备写数据

        Start();                                //发送起始命令
        SendData(0xa0);                         //发送设备地址+写命令
        RecvACK();
        SendData(0x00);                         //发送存储地址高字节
        RecvACK();
        SendData(0x00);                         //发送存储地址低字节
        RecvACK();
        Start();                                //发送起始命令
        SendData(0xa1);                         //发送设备地址+读命令
        RecvACK();
        P0 = RecvData();                        //读取数据 1
        SendACK();
        P2 = RecvData();                        //读取数据 2
        SendNAK();
        Stop();                                 //发送停止命令

        P_SW2 = 0x00;
```

STC MCU

```
    while (1);
}
```

汇编代码

```
;测试工作频率为11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
START:
            MOV     A,#00000001B        ;发送 START 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDDATA:
            MOV     DPTR,#I2CTXD        ;写数据到数据缓冲区
            MOVX    @DPTR,A
            MOV     A,#00000010B        ;发送 SEND 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVACK:
            MOV     A,#00000011B        ;发送读 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVDATA:
            MOV     A,#00000100B        ;发送 RECV 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
```

STC MCU

```
            CALL    WAIT
            MOV     DPTR,#I2CRXD        ;从数据缓冲区读取数据
            MOVX    A,@DPTR
            RET
SENDACK:
            MOV     A,#00000000B        ;设置 ACK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B        ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDNAK:
            MOV     A,#00000001B        ;设置 NAK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B        ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
STOP:
            MOV     A,#00000110B        ;发送 STOP 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
WAIT:
            MOV     DPTR,#I2CMSST       ;清中断标志
            MOVX    A,@DPTR
            JNB     ACC.6,WAIT
            ANL     A,#NOT 40H
            MOVX    @DPTR,A
            RET

DELAY:
            MOV     R0,#0
            MOV     R1,#0
DELAY1:
            NOP
            NOP
            NOP
            NOP
            DJNZ    R1,DELAY1
            DJNZ    R0,DELAY1
            RET

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H
```

STC MCU

```
MOV     P_SW2,#80H

MOV     A,#11100000B        ;设置 I2C 模块为主机模式
MOV     DPTR,#I2CCFG
MOVX    @DPTR,A
MOV     A,#00000000B
MOV     DPTR,#I2CMSST
MOVX    @DPTR,A

CALL    START               ;发送起始命令
MOV     A,#0A0H
CALL    SENDDATA            ;发送设备地址+写命令
CALL    RECVACK
MOV     A,#000H             ;发送存储地址高字节
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H             ;发送存储地址低字节
CALL    SENDDATA
CALL    RECVACK
MOV     A,#12H              ;写测试数据 1
CALL    SENDDATA
CALL    RECVACK
MOV     A,#78H              ;写测试数据 2
CALL    SENDDATA
CALL    RECVACK
CALL    STOP                ;发送停止命令

CALL    DELAY               ;等待设备写数据

CALL    START               ;发送起始命令
MOV     A,#0A0H             ;发送设备地址+写命令
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H             ;发送存储地址高字节
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H             ;发送存储地址低字节
CALL    SENDDATA
CALL    RECVACK
CALL    START               ;发送起始命令
MOV     A,#0A1H             ;发送设备地址+读命令
CALL    SENDDATA
CALL    RECVACK
CALL    RECVDATA            ;读取数据 1
MOV     P0,A
CALL    SENDACK
CALL    RECVDATA            ;读取数据 2
MOV     P2,A
CALL    SENDNAK
CALL    STOP                ;发送停止命令

JMP     $

END
```

20.4.3 I²C主机模式访问PCF8563

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P_SW2           =   0xba;

#define     I2CCFG          (*(unsigned char volatile xdata *)0xfe80)
#define     I2CMSCR         (*(unsigned char volatile xdata *)0xfe81)
#define     I2CMSST         (*(unsigned char volatile xdata *)0xfe82)
#define     I2CSLCR         (*(unsigned char volatile xdata *)0xfe83)
#define     I2CSLST         (*(unsigned char volatile xdata *)0xfe84)
#define     I2CSLADR        (*(unsigned char volatile xdata *)0xfe85)
#define     I2CTXD          (*(unsigned char volatile xdata *)0xfe86)
#define     I2CRXD          (*(unsigned char volatile xdata *)0xfe87)

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        SDA             =   P1^4;
sbit        SCL             =   P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                             //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                               //写数据到数据缓冲区
    I2CMSCR = 0x02;                             //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                             //发送读 ACK 命令
```

```
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                         //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                         //设置 ACK 信号
    I2CMSCR = 0x05;                         //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                         //设置 NAK 信号
    I2CMSCR = 0x05;                         //发送 ACK 命令
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                         //发送 STOP 命令
    Wait();
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;
```

STC MCU

```
    I2CCFG = 0xe0;                          //使能 I2C 主机模式
    I2CMSST = 0x00;

    Start();                                //发送起始命令
    SendData(0xa2);                         //发送设备地址+写命令
    RecvACK();
    SendData(0x02);                         //发送存储地址
    RecvACK();
    SendData(0x00);                         //设置秒值
    RecvACK();
    SendData(0x00);                         //设置分钟值
    RecvACK();
    SendData(0x12);                         //设置小时值
    RecvACK();
    Stop();                                 //发送停止命令

    while (1)
    {
        Start();                            //发送起始命令
        SendData(0xa2);                     //发送设备地址+写命令
        RecvACK();
        SendData(0x02);                     //发送存储地址
        RecvACK();
        Start();                            //发送起始命令
        SendData(0xa3);                     //发送设备地址+读命令
        RecvACK();
        P0 = RecvData();                    //读取秒值
        SendACK();
        P2 = RecvData();                    //读取分钟值
        SendACK();
        P3 = RecvData();                    //读取小时值
        SendNAK();
        Stop();                             //发送停止命令

        Delay();
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

P0M1        DATA    093H
P0M0        DATA    094H
```

```
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
START:
            MOV     A,#00000001B            ;发送 START 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDDATA:
            MOV     DPTR,#I2CTXD            ;写数据到数据缓冲区
            MOVX    @DPTR,A
            MOV     A,#00000010B            ;发送 SEND 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVACK:
            MOV     A,#00000011B            ;发送读 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVDATA:
            MOV     A,#00000100B            ;发送 RECV 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            CALL    WAIT
            MOV     DPTR,#I2CRXD            ;从数据缓冲区读取数据
            MOVX    A,@DPTR
            RET
SENDACK:
            MOV     A,#00000000B            ;设置 ACK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B            ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDNAK:
            MOV     A,#00000001B            ;设置 NAK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B            ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
STOP:
            MOV     A,#00000110B            ;发送 STOP 命令
            MOV     DPTR,#I2CMSCR
```

```
                MOVX    @DPTR,A
                JMP     WAIT
WAIT:
                MOV     DPTR,#I2CMSST           ;清中断标志
                MOVX    A,@DPTR
                JNB     ACC.6,WAIT
                ANL     A,#NOT 40H
                MOVX    @DPTR,A
                RET

DELAY:
                MOV     R0,#0
                MOV     R1,#0
DELAY1:
                NOP
                NOP
                NOP
                NOP
                DJNZ    R1,DELAY1
                DJNZ    R0,DELAY1
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                MOV     P_SW2,#80H

                MOV     A,#11100000B            ;设置 I2C 模块为主机模式
                MOV     DPTR,#I2CCFG
                MOVX    @DPTR,A
                MOV     A,#00000000B
                MOV     DPTR,#I2CMSST
                MOVX    @DPTR,A

                CALL    START                   ;发送起始命令
                MOV     A,#0A2H
                CALL    SENDDATA                ;发送设备地址+写命令
                CALL    RECVACK
                MOV     A,#002H                 ;发送存储地址
                CALL    SENDDATA
                CALL    RECVACK
                MOV     A,#00H                  ;设置秒值
                CALL    SENDDATA
                CALL    RECVACK
                MOV     A,#00H                  ;设置分钟值
                CALL    SENDDATA
                CALL    RECVACK
```

```
                MOV     A,#12H          ;设置小时值
                CALL    SENDDATA
                CALL    RECVACK
                CALL    STOP            ;发送停止命令
LOOP:
                CALL    START           ;发送起始命令
                MOV     A,#0A2H         ;发送设备地址+写命令
                CALL    SENDDATA
                CALL    RECVACK
                MOV     A,#002H         ;发送存储地址
                CALL    SENDDATA
                CALL    RECVACK
                CALL    START           ;发送起始命令
                MOV     A,#0A3H         ;发送设备地址+读命令
                CALL    SENDDATA
                CALL    RECVACK
                CALL    RECVDATA        ;读取秒值
                MOV     P0,A
                CALL    SENDACK
                CALL    RECVDATA        ;读取分钟值
                MOV     P2,A
                CALL    SENDACK
                CALL    RECVDATA        ;读取小时值
                MOV     P3,A
                CALL    SENDNAK
                CALL    STOP            ;发送停止命令

                CALL    DELAY

                JMP     LOOP

                END
```

20.4.4 I²C从机模式（中断方式）

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P0M1        =   0x93;
```

```
sfr         P0M0            =   0x94;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        SDA             =   P1^4;
sbit        SCL             =   P1^5;

bit         isda;                                   //设备地址标志
bit         isma;                                   //存储地址标志
unsigned char               addr;
unsigned char pdata         buffer[256];

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;

    if (I2CSLST & 0x40)
    {
        I2CSLST &= ~0x40;                           //处理 START 事件
    }
    else if (I2CSLST & 0x20)
    {
        I2CSLST &= ~0x20;                           //处理 RECV 事件
        if (isda)
        {
            isda = 0;                               //处理 RECV 事件（RECV DEVICE ADDR）
        }
        else if (isma)
        {
            isma = 0;                               //处理 RECV 事件（RECV MEMORY ADDR）
            addr = I2CRXD;
            I2CTXD = buffer[addr];
        }
        else
        {
            buffer[addr++] = I2CRXD;                //处理 RECV 事件（RECV DATA）
        }
    }
    else if (I2CSLST & 0x10)
    {
        I2CSLST &= ~0x10;                           //处理 SEND 事件
        if (I2CSLST & 0x02)
        {
            I2CTXD = 0xff;                          //接收到 NAK 则停止读取数据
        }
        else
        {
            I2CTXD = buffer[++addr];                //接收到 ACK 则继续读取数据
        }
    }
    else if (I2CSLST & 0x08)
    {
```

```
            I2CSLST &= ~0x08;                       //处理 STOP 事件
            isda = 1;
            isma = 1;
        }

        _pop_(P_SW2);
}

void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        P_SW2 = 0x80;

        I2CCFG = 0x81;                          //使能 I2C 从机模式
        I2CSLADR = 0x5a;                        //设置从机设备地址寄存器 I2CSLADR=0101_1010B
                                                //即 I2CSLADR[7:1]=010_1101B,MA=0B。
                                                //由于 MA 为 0,主机发送的的设备地址必须与
                                                //I2CSLADR[7:1]相同才能访问此 I2C 从机设备。
                                                //主机若需要写数据则要发送 5AH(0101_1010B)
                                                //主机若需要读数据则要发送 5BH(0101_1011B)
        I2CSLST = 0x00;
        I2CSLCR = 0x78;                         //使能从机模式中断
        EA = 1;

        isda = 1;                               //用户变量初始化
        isma = 1;
        addr = 0;
        I2CTXD = buffer[addr];

        while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H
```

```
SDA             BIT             P1.4
SCL             BIT             P1.5
ISDA            BIT             20H.0           ;设备地址标志
ISMA            BIT             20H.1           ;存储地址标志

ADDR            DATA            21H

P1M1            DATA            091H
P1M0            DATA            092H
P0M1            DATA            093H
P0M0            DATA            094H
P2M1            DATA            095H
P2M0            DATA            096H
P3M1            DATA            0B1H
P3M0            DATA            0B2H
P4M1            DATA            0B3H
P4M0            DATA            0B4H
P5M1            DATA            0C9H
P5M0            DATA            0CAH

                ORG             0000H
                LJMP            MAIN
                ORG             00C3H
                LJMP            I2CISR

                ORG             0100H
I2CISR:
                PUSH            ACC
                PUSH            PSW
                PUSH            DPL
                PUSH            DPH
                MOV             DPTR,#I2CSLST   ;检测从机状态
                MOVX            A,@DPTR
                JB              ACC.6,STARTIF
                JB              ACC.5,RXIF
                JB              ACC.4,TXIF
                JB              ACC.3,STOPIF
ISREXIT:
                POP             DPH
                POP             DPL
                POP             PSW
                POP             ACC
                RETI
STARTIF:
                ANL             A,#NOT 40H      ;处理 START 事件
                MOVX            @DPTR,A
                JMP             ISREXIT
RXIF:
                ANL             A,#NOT 20H      ;处理 RECV 事件
                MOVX            @DPTR,A
                MOV             DPTR,#I2CRXD
                MOVX            A,@DPTR
                JBC             ISDA,RXDA
                JBC             ISMA,RXMA
                MOV             R0,ADDR         ;处理 RECV 事件（RECV DATA）
                MOVX            @R0,A
                INC             ADDR
                JMP             ISREXIT
RXDA:
```

STC MCU

```
            JMP     ISREXIT             ;处理 RECV 事件（RECV DEVICE ADDR）
RXMA:
            MOV     ADDR,A              ;处理 RECV 事件（RECV MEMORY ADDR）
            MOV     R0,A
            MOVX    A,@R0
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     ISREXIT
TXIF:
            ANL     A,#NOT 10H          ;处理 SEND 事件
            MOVX    @DPTR,A
            JB      ACC.1,RXNAK
            INC     ADDR
            MOV     R0,ADDR
            MOVX    A,@R0
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     ISREXIT
RXNAK:
            MOVX    A,#0FFH
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     ISREXIT
STOPIF:
            ANL     A,#NOT 08H          ;处理 STOP 事件
            MOVX    @DPTR,A
            SETB    ISDA
            SETB    ISMA
            JMP     ISREXIT

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P_SW2,#80H

            MOV     A,#10000001B        ;使能 I2C 从机模式
            MOV     DPTR,#I2CCFG
            MOVX    @DPTR,A
            MOV     A,#01011010B        ;设置从机设备地址寄存器 I2CSLADR=0101_1010B
                                        ;即 I2CSLADR[7:1]=010_1101B,MA=0B。
                                        ;由于 MA 为 0,主机发送的的设备地址必须与
                                        ;I2CSLADR[7:1]相同才能访问此 I2C 从机设备。
                                        ;主机若需要写数据则要发送 5AH(0101_1010B)
                                        ;主机若需要读数据则要发送 5BH(0101_1011B)
            MOV     DPTR,#I2CSLADR
            MOVX    @DPTR,A
            MOV     A,#00000000B
```

```
        MOV     DPTR,#I2CSLST
        MOVX    @DPTR,A
        MOV     A,#01111000B            ;使能从机模式中断
        MOV     DPTR,#I2CSLCR
        MOVX    @DPTR,A

        SETB    ISDA                    ;用户变量初始化
        SETB    ISMA
        CLR     A
        MOV     ADDR,A
        MOV     R0,A
        MOVX    A,@R0
        MOV     DPTR,#I2CTXD
        MOVX    @DPTR,A

        SETB    EA

        SJMP    $

        END
```

20.4.5 I²C从机模式（查询方式）

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sbit    SDA         =   P1^4;
sbit    SCL         =   P1^5;
```

```
bit             isda;                                   //设备地址标志
bit             isma;                                   //存储地址标志
unsigned char               addr;
unsigned char pdata         buffer[256];

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0x81;                                      //使能 I2C 从机模式
    I2CSLADR = 0x5a;                                    //设置从机设备地址寄存器 I2CSLADR=0101_1010B
                                                        //即 I2CSLADR[7:1]=010_1101B,MA=0B。
                                                        //由于 MA 为 0,主机发送的的设备地址必须与
                                                        //I2CSLADR[7:1]相同才能访问此 I2C 从机设备。
                                                        //主机若需要写数据则要发送 5AH(0101_1010B)
                                                        //主机若需要读数据则要发送 5BH(0101_1011B)
    I2CSLST = 0x00;
    I2CSLCR = 0x00;                                     //禁止从机模式中断

    isda = 1;                                           //用户变量初始化
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1)
    {
        if (I2CSLST & 0x40)
        {
            I2CSLST &= ~0x40;                           //处理 START 事件
        }
        else if (I2CSLST & 0x20)
        {
            I2CSLST &= ~0x20;                           //处理 RECV 事件
            if (isda)
            {
                isda = 0;                               //处理 RECV 事件（RECV DEVICE ADDR）
            }
            else if (isma)
            {
                isma = 0;                               //处理 RECV 事件（RECV MEMORY ADDR）
                addr = I2CRXD;
                I2CTXD = buffer[addr];
            }
            else
            {
```

```
                buffer[addr++] = I2CRXD;                //处理 RECV 事件（RECV DATA）
            }
        }
        else if (I2CSLST & 0x10)
        {
            I2CSLST &= ~0x10;                           //处理 SEND 事件
            if (I2CSLST & 0x02)
            {
                I2CTXD = 0xff;                          //接收到 NAK 则停止读取数据
            }
            else
            {
                I2CTXD = buffer[++addr];                //接收到 ACK 则继续读取数据
            }
        }
        else if (I2CSLST & 0x08)
        {
            I2CSLST &= ~0x08;                           //处理 STOP 事件
            isda = 1;
            isma = 1;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5
ISDA        BIT     20H.0               ;设备地址标志
ISMA        BIT     20H.1               ;存储地址标志

ADDR        DATA    21H

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH
```

```
        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#80H

        MOV     A,#10000001B            ;使能 I2C 从机模式
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#01011010B            ;设置从机设备地址寄存器 I2CSLADR=0101_1010B
                                        ;即 I2CSLADR[7:1]=010_1101B,MA=0B。
                                        ;由于 MA 为 0,主机发送的的设备地址必须与
                                        ;I2CSLADR[7:1]相同才能访问此 I2C 从机设备。
                                        ;主机若需要写数据则要发送 5AH(0101_1010B)
                                        ;主机若需要读数据则要发送 5BH(0101_1011B)
        MOV     DPTR,#I2CSLADR
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CSLST
        MOVX    @DPTR,A
        MOV     A,#00000000B            ;禁止从机模式中断
        MOV     DPTR,#I2CSLCR
        MOVX    @DPTR,A

        SETB    ISDA                    ;用户变量初始化
        SETB    ISMA
        CLR     A
        MOV     ADDR,A
        MOV     R0,A
        MOVX    A,@R0
        MOV     DPTR,#I2CTXD
        MOVX    @DPTR,A

LOOP:
        MOV     DPTR,#I2CSLST           ;检测从机状态
        MOVX    A,@DPTR
        JB      ACC.6,STARTIF
        JB      ACC.5,RXIF
        JB      ACC.4,TXIF
        JB      ACC.3,STOPIF
        JMP     LOOP
STARTIF:
        ANL     A,#NOT 40H              ;处理 START 事件
        MOVX    @DPTR,A
```

```
            JMP     LOOP
RXIF:
            ANL     A,#NOT 20H          ;处理 RECV 事件
            MOVX    @DPTR,A
            MOV     DPTR,#I2CRXD
            MOVX    A,@DPTR
            JBC     ISDA,RXDA
            JBC     ISMA,RXMA
            MOV     R0,ADDR             ;处理 RECV 事件（RECV DATA）
            MOVX    @R0,A
            INC     ADDR
            JMP     LOOP
RXDA:
            JMP     LOOP                ;处理 RECV 事件（RECV DEVICE ADDR）
RXMA:
            MOV     ADDR,A              ;处理 RECV 事件（RECV MEMORY ADDR）
            MOV     R0,A
            MOVX    A,@R0
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     LOOP
TXIF:
            ANL     A,#NOT 10H          ;处理 SEND 事件
            MOVX    @DPTR,A
            JB      ACC.1,RXNAK
            INC     ADDR
            MOV     R0,ADDR
            MOVX    A,@R0
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     LOOP
RXNAK:
            MOVX    A,#0FFH
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     LOOP
STOPIF:
            ANL     A,#NOT 08H          ;处理 STOP 事件
            MOVX    @DPTR,A
            SETB    ISDA
            SETB    ISMA
            JMP     LOOP

            END
```

20.4.6 测试I²C从机模式代码的主机代码

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P_SW2           =     0xba;
```

```
#define   I2CCFG        (*(unsigned char volatile xdata *)0xfe80)
#define   I2CMSCR       (*(unsigned char volatile xdata *)0xfe81)
#define   I2CMSST       (*(unsigned char volatile xdata *)0xfe82)
#define   I2CSLCR       (*(unsigned char volatile xdata *)0xfe83)
#define   I2CSLST       (*(unsigned char volatile xdata *)0xfe84)
#define   I2CSLADR      (*(unsigned char volatile xdata *)0xfe85)
#define   I2CTXD        (*(unsigned char volatile xdata *)0xfe86)
#define   I2CRXD        (*(unsigned char volatile xdata *)0xfe87)

sfr       P1M1    =   0x91;
sfr       P1M0    =   0x92;
sfr       P0M1    =   0x93;
sfr       P0M0    =   0x94;
sfr       P2M1    =   0x95;
sfr       P2M0    =   0x96;
sfr       P3M1    =   0xb1;
sfr       P3M0    =   0xb2;
sfr       P4M1    =   0xb3;
sfr       P4M0    =   0xb4;
sfr       P5M1    =   0xc9;
sfr       P5M0    =   0xca;

sbit      SDA     =   P1^4;
sbit      SCL     =   P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                 //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                   //写数据到数据缓冲区
    I2CMSCR = 0x02;                 //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                 //发送读 ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                 //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
```

```
    I2CMSST = 0x00;                         //设置 ACK 信号
    I2CMSCR = 0x05;                         //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                         //设置 NAK 信号
    I2CMSCR = 0x05;                         //发送 ACK 命令
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                         //发送 STOP 命令
    Wait();
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0xe0;                          //使能 I2C 主机模式
    I2CMSST = 0x00;

    Start();                                //发送起始命令
    SendData(0x5a);                         //发送设备地址(010_1101B)+写命令(0B)
    RecvACK();
    SendData(0x00);                         //发送存储地址
    RecvACK();
    SendData(0x12);                         //写测试数据 1
    RecvACK();
    SendData(0x78);                         //写测试数据 2
```

STC MCU

```
    RecvACK();
    Stop();                                 //发送停止命令

    Start();                                //发送起始命令
    SendData(0x5a);                         //发送设备地址(010_1101B)+写命令(0B)
    RecvACK();
    SendData(0x00);                         //发送存储地址高字节
    RecvACK();
    Start();                                //发送起始命令
    SendData(0x5b);                         //发送设备地址(010_1101B)+读命令(1B)
    RecvACK();
    P0 = RecvData();                        //读取数据 1
    SendACK();
    P2 = RecvData();                        //读取数据 2
    SendNAK();
    Stop();                                 //发送停止命令

    P_SW2 = 0x00;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
START:
            MOV     A,#00000001B        ;发送 START 命令
```

```
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
SENDDATA:
                MOV     DPTR,#I2CTXD        ;写数据到数据缓冲区
                MOVX    @DPTR,A
                MOV     A,#00000010B        ;发送 SEND 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
RECVACK:
                MOV     A,#00000011B        ;发送读 ACK 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
RECVDATA:
                MOV     A,#00000100B        ;发送 RECV 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                CALL    WAIT
                MOV     DPTR,#I2CRXD        ;从数据缓冲区读取数据
                MOVX    A,@DPTR
                RET
SENDACK:
                MOV     A,#00000000B        ;设置 ACK 信号
                MOV     DPTR,#I2CMSST
                MOVX    @DPTR,A
                MOV     A,#00000101B        ;发送 ACK 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
SENDNAK:
                MOV     A,#00000001B        ;设置 NAK 信号
                MOV     DPTR,#I2CMSST
                MOVX    @DPTR,A
                MOV     A,#00000101B        ;发送 ACK 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
STOP:
                MOV     A,#00000110B        ;发送 STOP 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
WAIT:
                MOV     DPTR,#I2CMSST       ;清中断标志
                MOVX    A,@DPTR
                JNB     ACC.6,WAIT
                ANL     A,#NOT 40H
                MOVX    @DPTR,A
                RET

DELAY:
                MOV     R0,#0
                MOV     R1,#0
DELAY1:
                NOP
                NOP
                NOP
```

```
        NOP
        DJNZ    R1,DELAY1
        DJNZ    R0,DELAY1
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#80H

        MOV     A,#11100000B        ;设置 I2C 模块为主机模式
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A

        CALL    START               ;发送起始命令
        MOV     A,#5AH              ;
        CALL    SENDDATA            ;发送设备地址(010_1101B)+写命令(0B)
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#12H              ;写测试数据 1
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#78H              ;写测试数据 2
        CALL    SENDDATA
        CALL    RECVACK
        CALL    STOP                ;发送停止命令

        CALL    DELAY               ;等待设备写数据

        CALL    START               ;发送起始命令
        MOV     A,#5AH              ;发送设备地址(010_1101B)+写命令(0B)
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址
        CALL    SENDDATA
        CALL    RECVACK
        CALL    START               ;发送起始命令
        MOV     A,#5BH              ;发送设备地址(010_1101B)+读命令(1B)
        CALL    SENDDATA
        CALL    RECVACK
        CALL    RECVDATA            ;读取数据 1
        MOV     P0,A
```

```
        CALL    SENDACK
        CALL    RECVDATA            ;读取数据 2
        MOV     P2,A
        CALL    SENDNAK
        CALL    STOP                ;发送停止命令

        JMP     $

        END
```

STC MCU

21 增强型双数据指针

STC8A/8F 系列的单片机内部集成了两组 16 位的数据指针。通过程序控制, 可实现数据指针自动递增或递减功能以及两组数据指针的自动切换功能

21.1 相关的特殊功能寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DPL	数据指针（低字节）	82H									0000,0000
DPH	数据指针（高字节）	83H									0000,0000
DPL1	第二组数据指针（低字节）	E4H									0000,0000
DPH1	第二组数据指针（高字节）	E5H									0000,0000
DPS	DPTR 指针选择器	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
TA	DPTR 时序控制寄存器	AEH									0000,0000

21.1.1 第 1 组 16 位数据指针寄存器（DPTR0）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPL	82H								
DPH	83H								

DPL为低8位数据（低字节）

DPH为高8位数据（高字节）

DPL和DPH组合为第一组16位数据指针寄存器DPTR0

21.1.2 第 2 组 16 位数据指针寄存器（DPTR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPL1	E4H								
DPH1	E5H								

DPL1为低8位数据（低字节）

DPH1为高8位数据（高字节）

DPL1和DPH1组合为第二组16位数据指针寄存器DPTR1

21.1.3 数据指针控制寄存器（DPS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPS	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL

ID1：控制DPTR1自动递增方式

0：DPTR1 自动递增

1：DPTR1 自动递减

ID0：控制DPTR0自动递增方式

0：DPTR0 自动递增

1：DPTR0 自动递减

TSL：DPTR0/DPTR1自动切换控制（自动对SEL进行取反）

0：关闭自动切换功能

1：使能自动切换功能

当 TSL 位被置 1 后，每当执行完成相关指令后，系统会自动将 SEL 位取反。

与 TSL 相关的指令包括如下指令：

```
MOV     DPTR,#data16
INC     DPTR
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
```

AU1/AU0：使能DPTR1/DPTR0使用ID1/ID0控制位进行自动递增/递减控制

0：关闭自动递增/递减功能

1：使能自动递增/递减功能

注意：在写保护模式下，AU0 和 AU1 位无法直接单独使能，若单独使能 AU1 位，则 AU0 位也会被自动使能，若单独使能 AU0，没有效果。若需要单独使能 AU1 或者 AU0，则必须使用 TA 寄存器触发 DPS 的保护机制（参考 TA 寄存器的说明）。另外，只有执行下面的 3 条指令后才会对 DPTR0/DPTR1 进行自动递增/递减操作。3 条相关指令如下：

```
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
```

SEL：选择DPTR0/DPTR1作为当前的目标DPTR

0：选择 DPTR0 作为目标 DPTR

1：选择 DPTR1 作为目标 DPTR

SEL 选择目标 DPTR 对下面指令有效：

```
MOV     DPTR,#data16
INC     DPTR
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
JMP     @A+DPTR
```

21.1.4 数据指针控制寄存器（TA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TA	AEH								

TA寄存器是对DPS寄存器中的AU1和AU0进行写保护的。由于程序无法对DPS中的AU1和AU0进行单独的写入，所以当需要单独使能AU1或者AU0时，必须使用TA寄存器进行触发。TA寄存器是只写寄存器。当需要对AU1或者AU0进行单独使能时，必须按照如下的步骤进行操作：

```
CLR     EA          ;关闭中断（必需）
MOV     TA,#0AAH    ;写入触发命令序列 1
                    ;此处不能有其他任何指令
```

```
MOV     TA,#55H         ;写入触发命令序列 2
                        ;此处不能有其他任何指令
MOV     DPS,#xxH        ;写保护暂时关闭，可向 DPS 中写入任何值
                        ;DSP 再次进行写保护状态
SETB    EA              ;打开中断（如有必要）
```

STC MCU

21.2 范例程序

21.2.1 示例代码 1

将程序空间 1000H～1003H 的 4 个字节数据反向复制到扩展 RAM 的 0100H～0103H 中，即

C:1000H -> X:0103H
C:1001H -> X:0102H
C:1002H -> X:0101H
C:1003H -> X:0100H

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     DPS,#00100000B      ;使能 TSL,并选择 DPTR0
            MOV     DPTR,#1000H         ;将 1000H 写入 DPTR0 后选择 DPTR1 为 DPTR
            MOV     DPTR,#0103H         ;将 0103H 写入 DPTR1 中
            MOV     DPS,#10111000B      ;设置 DPTR1 为递减模式,DPTR0 为递加模式,使能 TSL
                                        ;AU0 和 AU1,并选择 DPTR0 为当前的 DPTR
            MOV     R7,#4               ;设置数据复制个数
COPY_NEXT:
            CLR     A                   ;
            MOVC    A,@A+DPTR           ;从 DPTR0 所指的程序空间读取数据,
                                        ;完成后 DPTR0 自动加 1 并将 DPTR1 设置为 DPTR
            MOVX    @DPTR,A             ;将 ACC 的数据写入到 DPTR1 所指的 XDATA 中,
```

```
                                    ;完成后 DPTR1 自动减 1 并将 DPTR0 设置为 DPTR
            DJNZ    R7,COPY_NEXT    ;

            SJMP    $

            END
```

21.2.2 示例代码 2

将扩展 RAM 的 0100H～0103H 中的数据依次发送到 P0 口

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            CLR     EA              ;关闭中断
            MOV     TA,#0AAH        ;写入 DPS 写保护触发命令 1
            MOV     TA,#55H         ;写入 DPS 写保护触发命令 2
            MOV     DPS,#00001000B  ;DPTR0 递增,单独使能 AU0,并选择 DPTR0
            SETB    EA              ;打开中断
            MOV     DPTR,#0100H     ;将 0100H 写入 DPTR0 中
            MOVX    A,@DPTR         ;从 DPTR0 所指的 XRAM 读取数据后 DPTR0 自动加 1
            MOV     P0,A            ;数据输出到 P0 口
            MOVX    A,@DPTR         ;从 DPTR0 所指的 XRAM 读取数据后 DPTR0 自动加 1
            MOV     P0,A            ;数据输出到 P0 口
            MOVX    A,@DPTR         ;从 DPTR0 所指的 XRAM 读取数据后 DPTR0 自动加 1
            MOV     P0,A            ;数据输出到 P0 口
            MOVX    A,@DPTR         ;从 DPTR0 所指的 XRAM 读取数据后 DPTR0 自动加 1
```

```
MOV     P0,A            ;数据输出到 P0 口
SJMP    $
END
```

STC MCU

附录A 编译器（汇编器）/仿真器使用指南

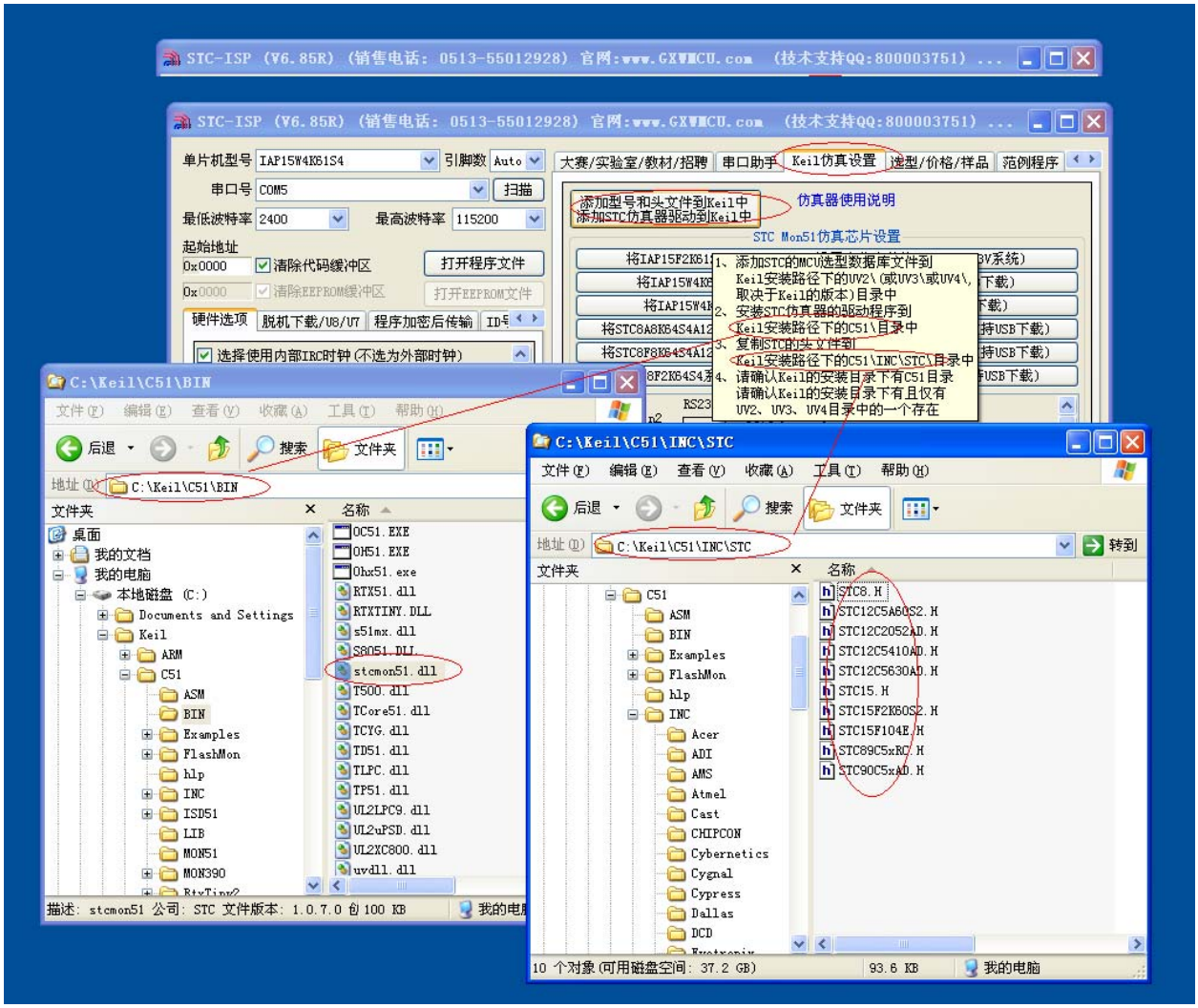
A：STC 单片机应使用何种编译器/汇编器？

Q：任何老式的 8051 编译器/汇编器都可以支持，现流行使用 Keil C51

A：Keil 环境中，应如何包含头文件

Q：按照下面图示的步骤安装完驱动和头文件后，新建项目时选择 STC 相应的单片机型号，在源文件中直接使用“#include <stc8g.h>”即可完成头文件的包含。如果新建项目时选择的 Intel 的 8052/87C52/87C54/87C58 或 Philips 的 P87C52/P87C54/P87C58 编译，头文件包含<reg51.h>即可，不过 STC 新增的特殊功能寄存器则需要用户自己声明。

1、安装 Keil 版本的仿真驱动



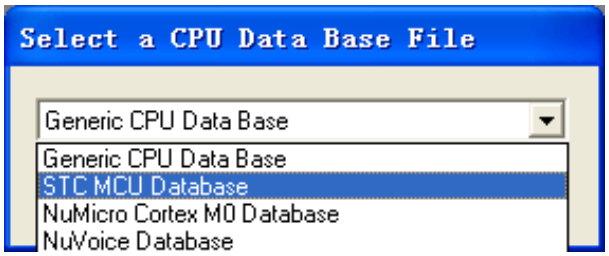
如上图，首先选择“Keil 仿真设置”页面，点击“添加 MCU 型号到 Keil 中”，在出现的如下的目录选择窗口中，定位到 Keil 的安装目录(一般可能为“C:\Keil\”)，“确定”后出现下图中右边所示的提示信息，表示安装成功。添加头文件的同时也会安装 STC 的 Monitor51 仿真驱动 STCMON51.DLL，驱动与头文件的的安装目录如上图所示。



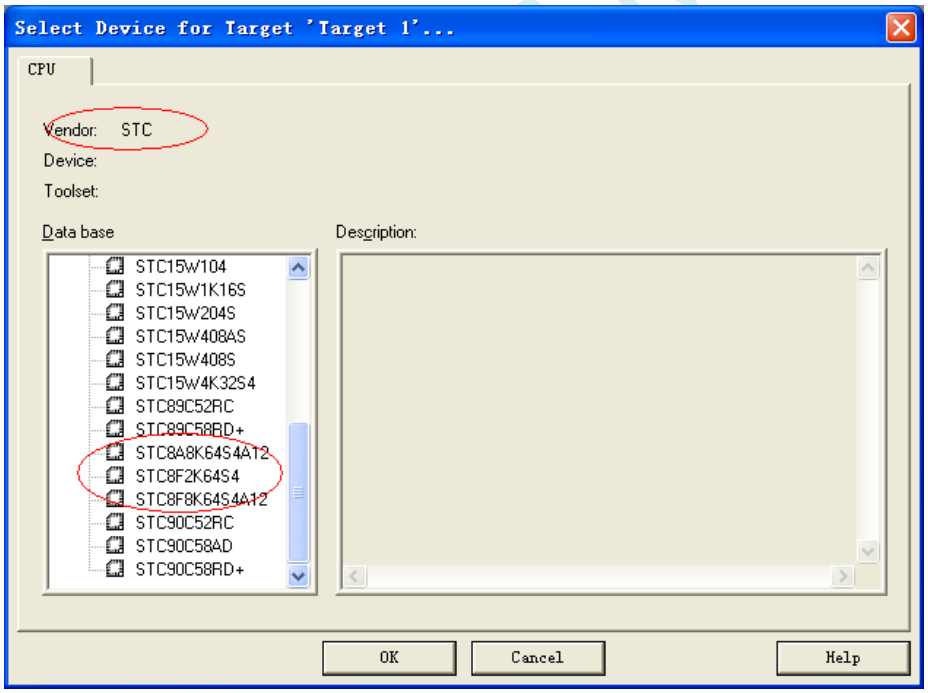


2、在 Keil 中创建项目

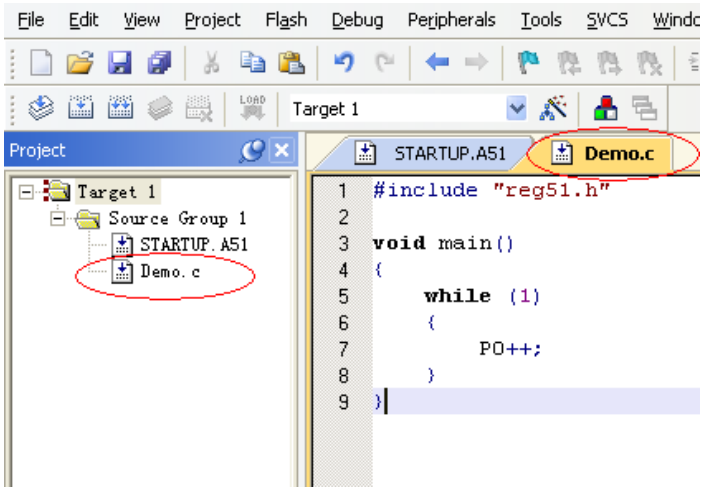
若第一步的驱动安装成功，则在 Keil 中新建项目时选择芯片型号时，便会有“STC MCU Database”的选择项，如下图



然后从列表中选择响应的 MCU 型号，我们在此选择“STC8A8K64S4A12”的型号，点击“确定”完成选择



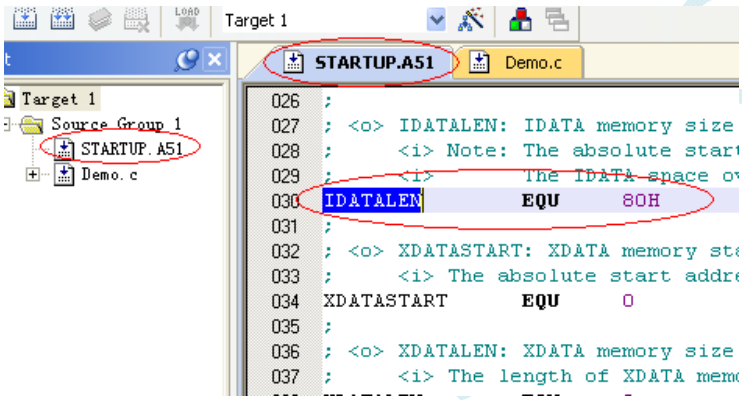
添加源代码文件到项目中，如下图：



保存项目，若编译无误，则可以进行下面的项目设置了

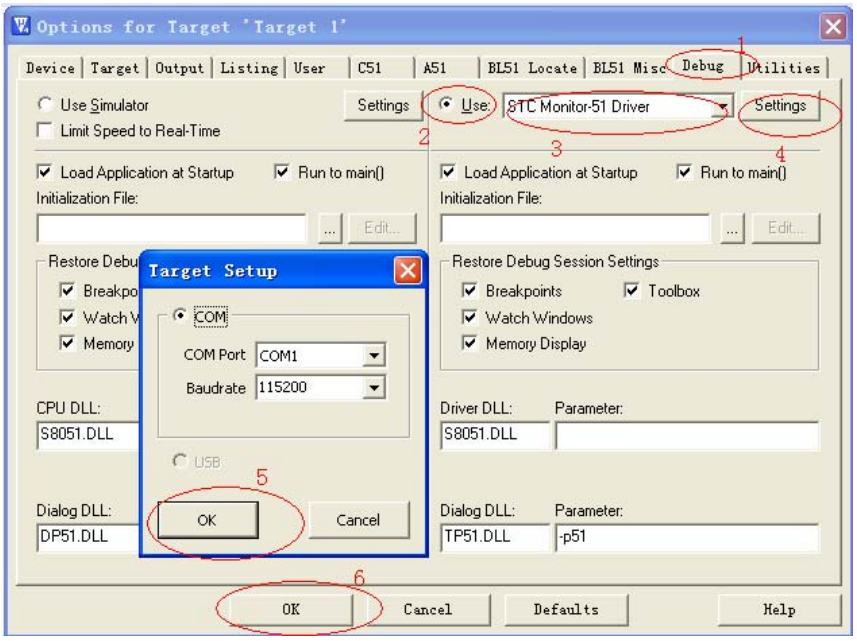
附加说明一点：

当创建的是 C 语言项目，且有将启动文件“STARTUP.A51”添加到项目中时，里面有一个命名为“IDATALEN”的宏定义，它是用来定义 IDATA 大小的一个宏，默认值是 128，即十六进制的 80H，同时它也是启动文件中需要初始化为 0 的 IDATA 的大小。所以当 IDATA 定义为 80H，那么 STARTUP.A51 里面的代码则会将 IDATA 的 00-7F 的 RAM 初始化为 0；同样若将 IDATA 定义为 0FFH，则会将 IDATA 的 00-FF 的 RAM 初始化为 0。



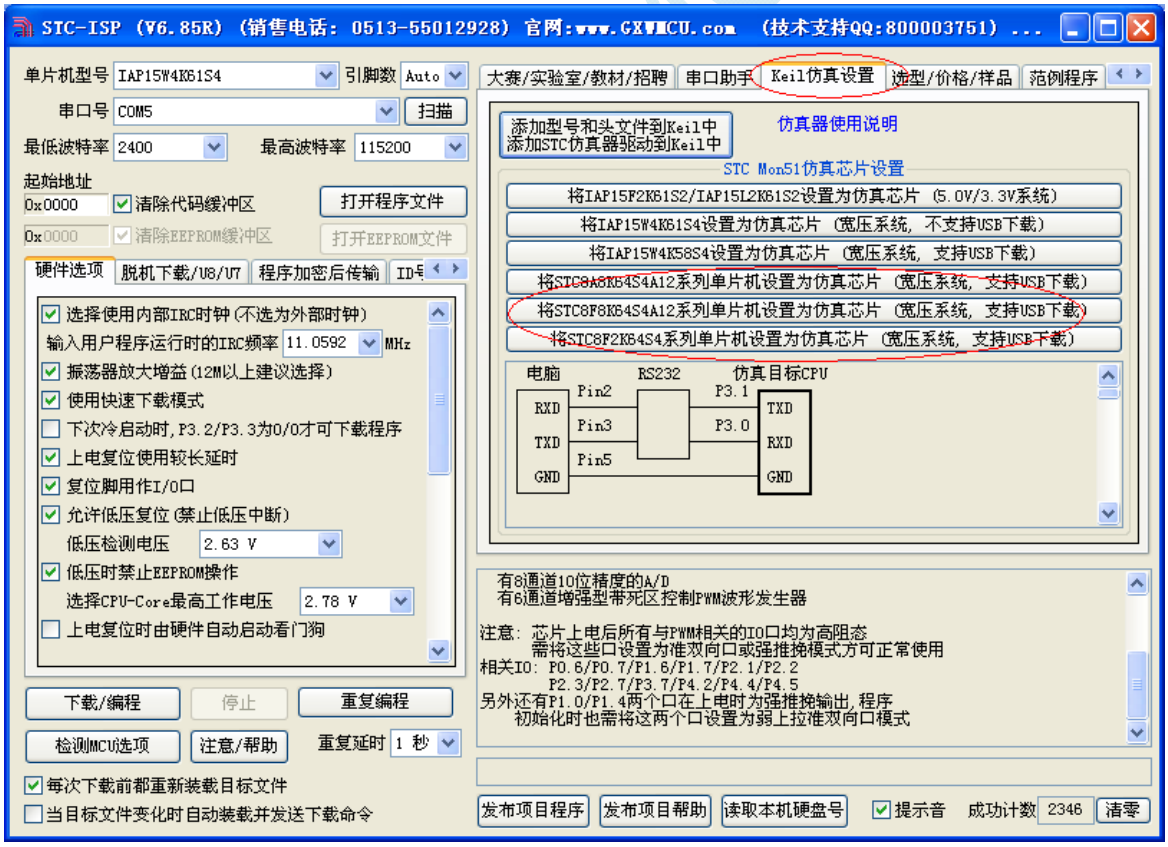
虽然 STC8 系列的单片机的 IDATA 大小为 256 字节（00-7F 的 DATA 和 80H-FFH 的 IDATA），但由于在 RAM 的最后 17 个字节有写入 ID 号以及相关的测试参数，若用户在程序中需要使用这一部分数据，则一定不要将 IDATALEN 定义为 256。

3、项目设置，选择 STC 仿真驱动



如上图，首先进入到项目的设置页面，选择“Debug”设置页，第 2 步选择右侧的硬件仿真“Use …”，第 3 步，在仿真驱动下拉列表中选择“STC Monitor-51 Driver”项，然后点击“Settings”按钮，进入下面的设置画面，对串口的端口号和波特率进行设置，波特率一般选择 115200。到此设置便完成了。

4、创建仿真芯片



准备一颗 STC8A 系列或者 STC8F 系列的芯片，并通过下载板连接到电脑的串口，然后如上图，选择正确的芯片型号，然后进入到“Keil 仿真设置”页面，点击相应型号的按钮，当程序下载完成后仿真器便制作完成了。

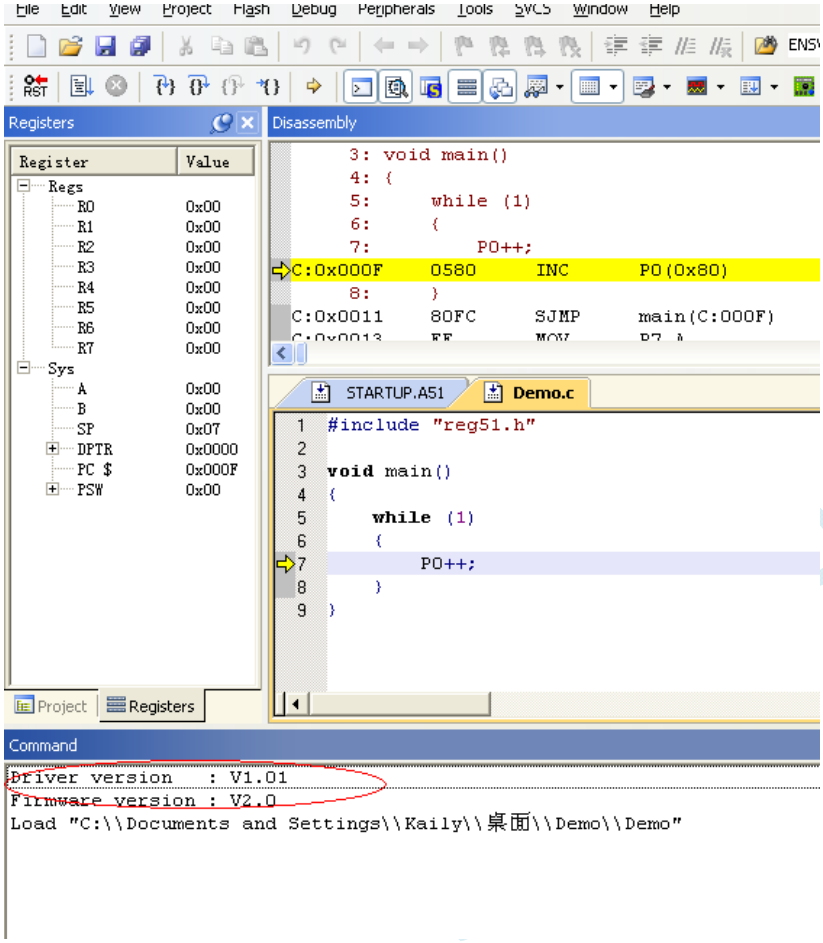
5、开始仿真

将制作完成的仿真芯片通过串口与电脑相连接。

将前面我们所创建的项目编译至没有错误后，按“Ctrl+F5”开始调试。

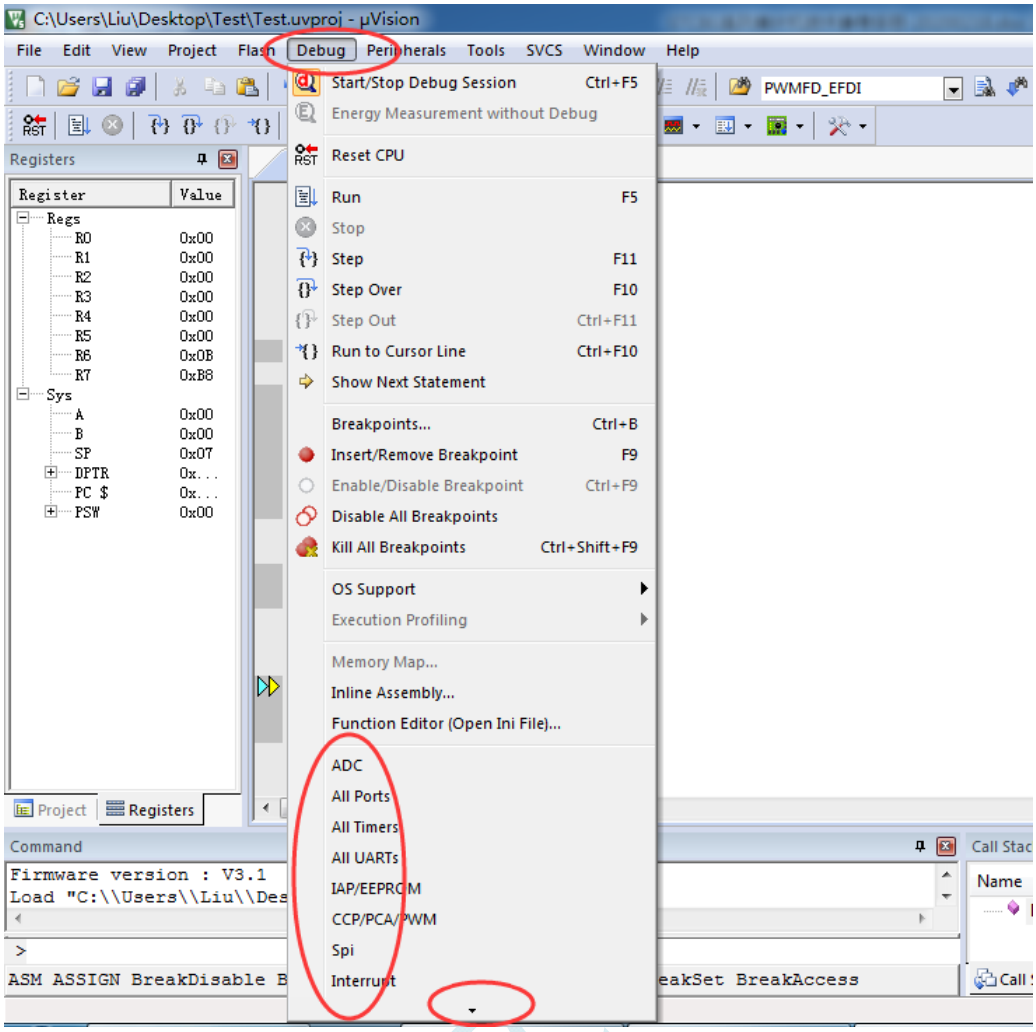
若硬件连接无误的话，将会进入到类似于下面的调试界面，并在命令输出窗口显示当前的仿真驱动版本号和当前仿真监控代码固件的版本号

断点设置的个数目前最大允许 20 个（理论上可设置任意个，但是断点设置得过多会影响调试的速度）。



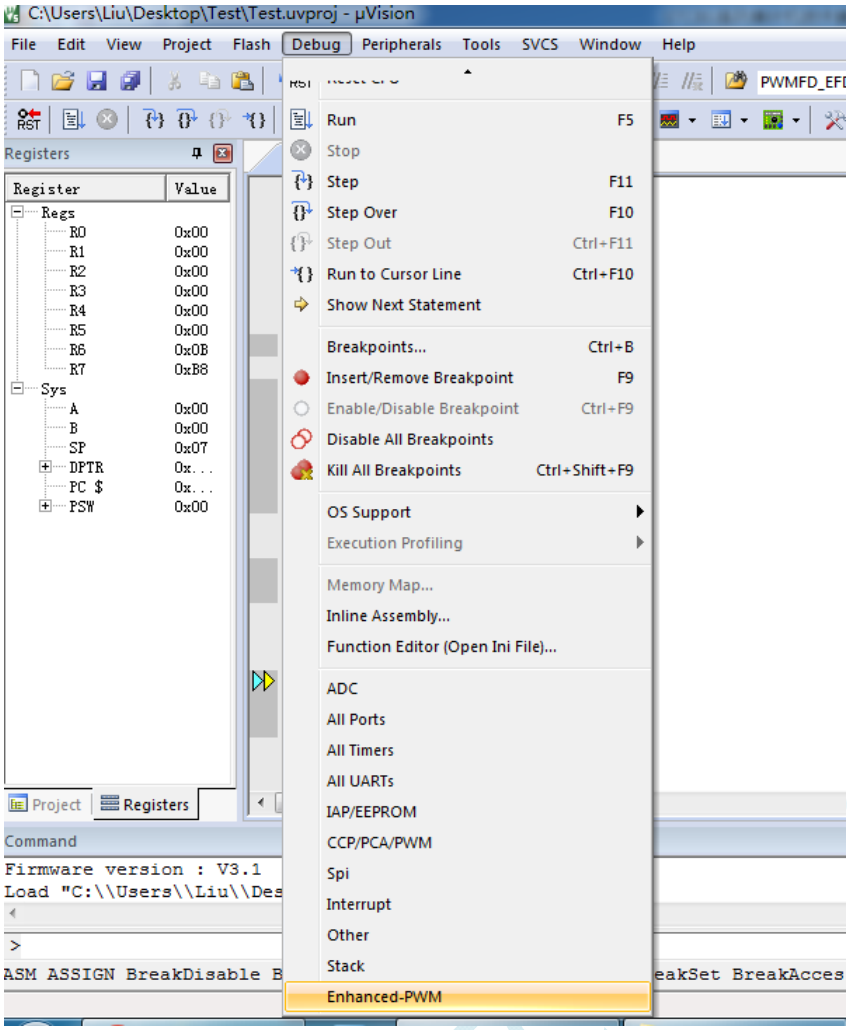
6、仿真过程中，寄存器的查看

在仿真的过程中，可查看 MCU 相关的寄存器。所有的寄存器列表在“Debug”菜单的底端。如下图所示：



在上图“Debug”菜单的最底端，还有一个黑色的小三角，这表示还有隐藏的项目（主要是由于显示版面大小的原因）

将鼠标仿真小三角上即可自动拖出所有的项目，如下图：



仿真注意事项：

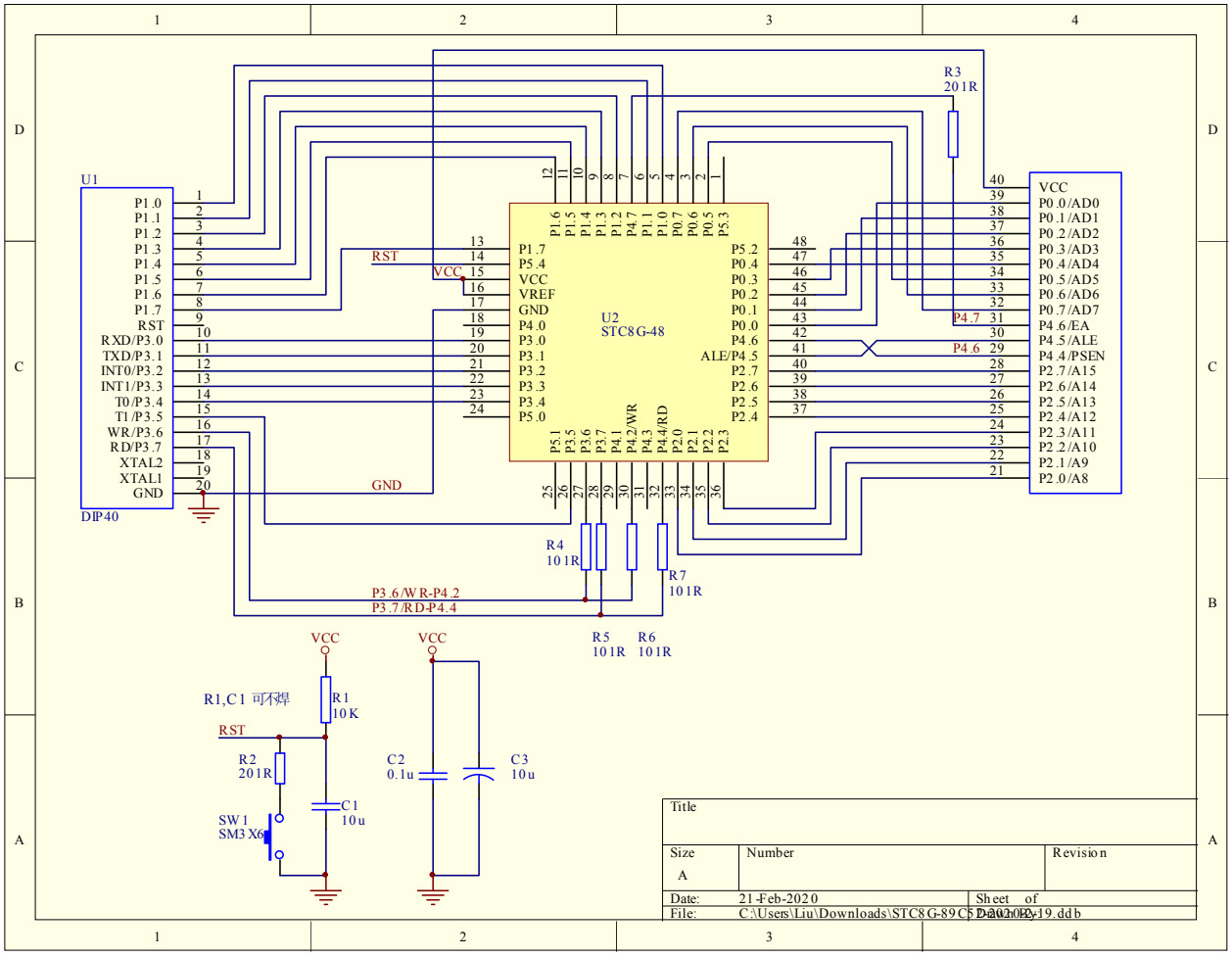
1、 仿真监控程序占用 P3.0/P3.1 两个端口，但不占用串口 1，用户可以将串口 1 切换到 P3.6/P3.7 或者 P1.6/P1.7 再使用

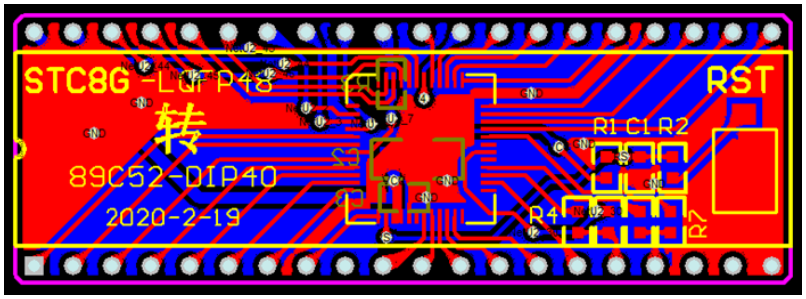
2、 仿真监控程序占用内部扩展 RAM(XDATA)的最后 768 字节，用户不可对这个区域的 XDATA 进行写操作

附录B 如何让传统的 8051 单片机学习板可仿真

传统的 8051 单片机学习板不具有仿真功能，让传统的 8051 单片机学习板可仿真需要借助转换板，转换板的实物图如下图所示，转换后的引脚排布与传统 8051 的脚位基本一致，从而可以实现标准 8051 学习板的仿真功能。

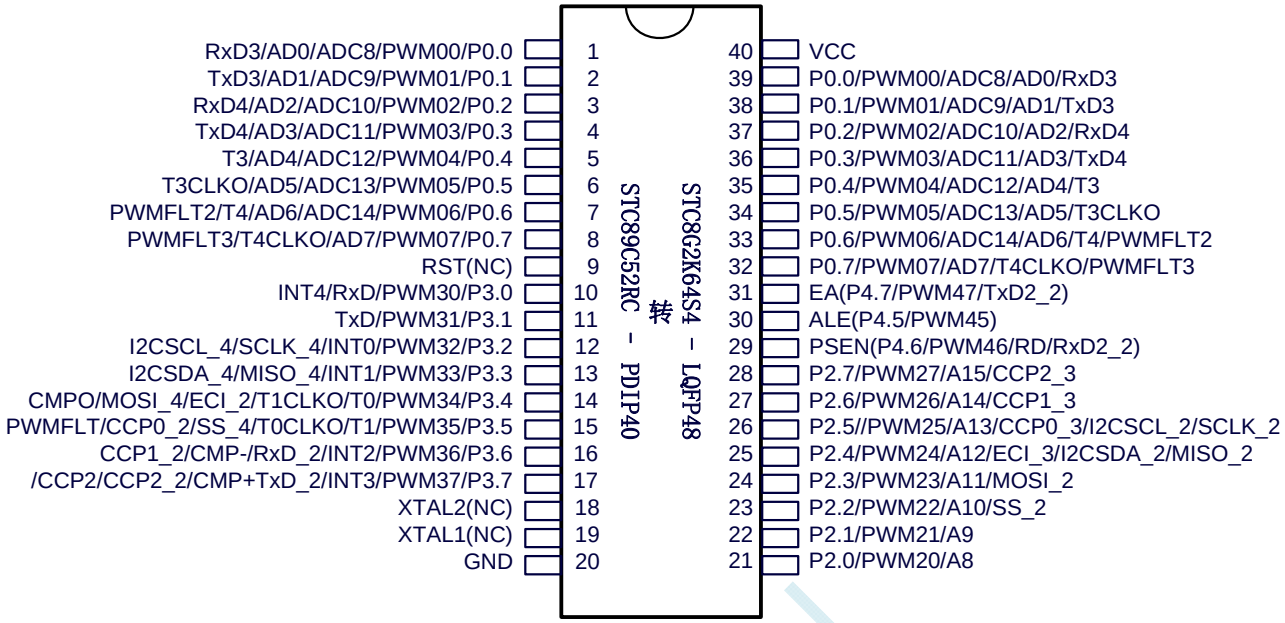
下图是转换板的原理图和 PCB 板图





该转换板可进行 STC8G 系列 LQFP48 转 STC89C52RC/STC89C58RD+系列仿真用。

下图为转换板功能示意图



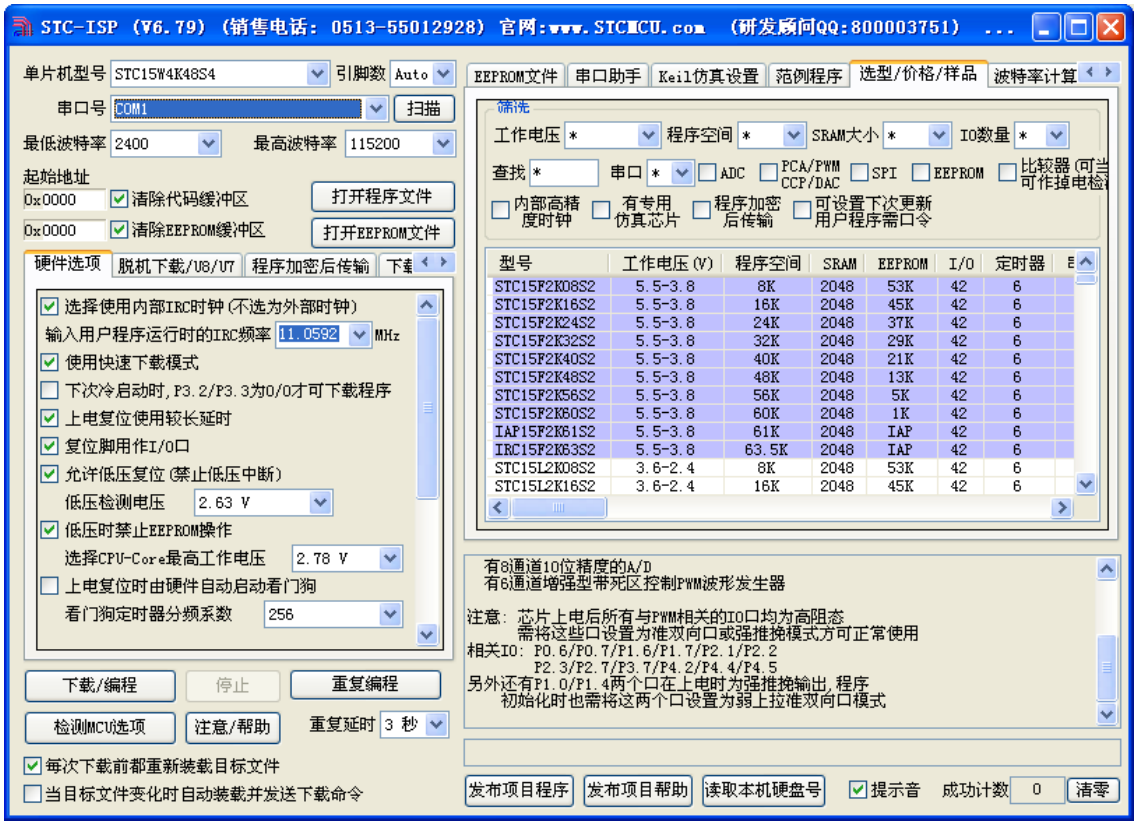
注意：

- ✓ 由于内置高精准 R/C 时钟，因此不需要外部晶振,XTAL1 和 XTAL2 是空的
- ✓ WR 和 RD 是(WR/P4.2 和 RD/P4.4)，而不是传统的(WR/P3.6 和 RD/P3.7)。
  （转换板中，P4.2 与 P3.6 连接在一起，P4.4 与 P3.7 连接在一起。当用户需要用此转换板访问外部总线时，需要将 P3.6 和 P3.7 设置为高阻输入模式，从而使 P4.2 和 P4.4 正常输出总线读写信号；若不需要访问外部总线，则需将 P4.2 和 P4.4 设置高阻输入模式，3.6 和 P3.7 即为普通 I/O。）
- ✓ 由于 STC8G 系列 MCU 是低电平复位，与传统 8051 的高电平复位不兼容，因此 RST 管脚是悬空，而用转换板上的复位按键加复位电路取代

附录C STC-USB驱动程序安装说明

Windows XP 安装方法

打开 V6.79 版（或者更新的版本）的 STC-ISP 下载软件，下载软件会自动将驱动文件复制到相关的系统目录



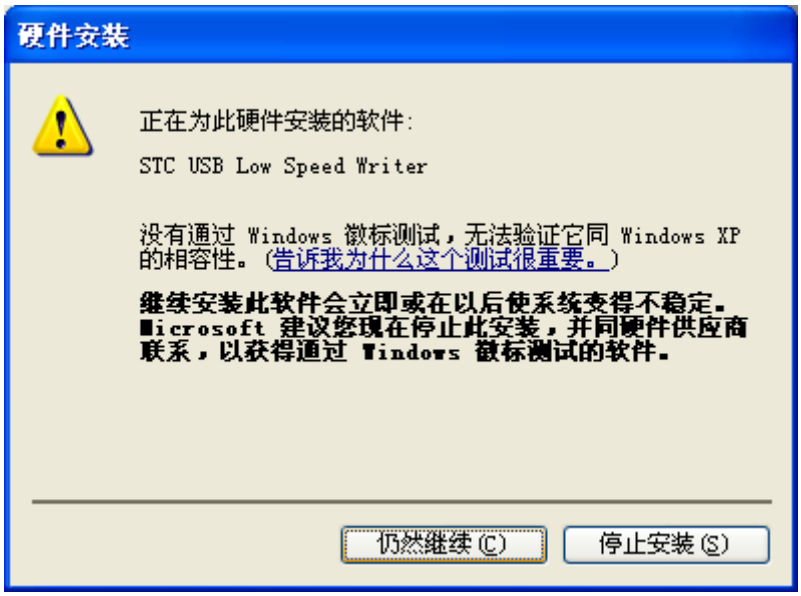
插入 USB 设备，系统找到设备后自动弹出如下对话框，选择其中的“否，暂时不”项



在下面的对话框中选择“自动安装软件(推荐)”项



在弹出的下列对话框中，选择“仍然继续”按钮



STC MCU

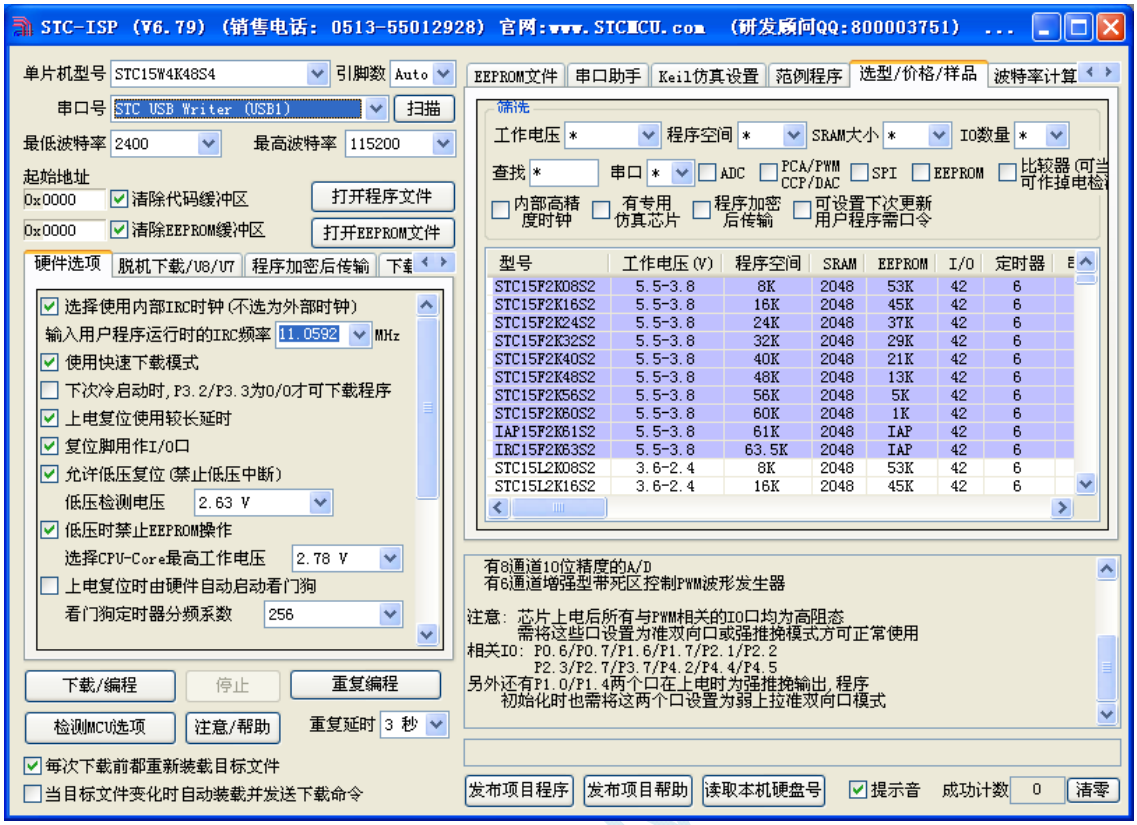
接下系统会自动安装驱动，如下图



出现下面的的对话框表示驱动安装完成

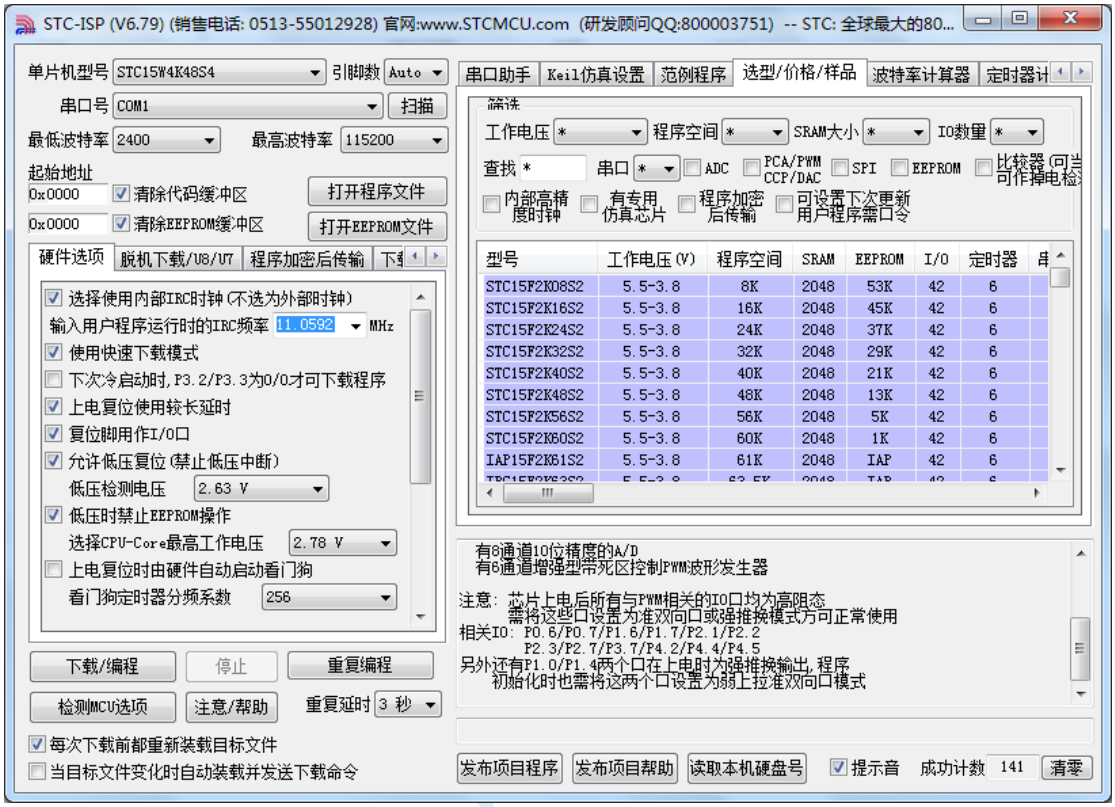


此时，之前打开的 STC-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)” ，如下图：

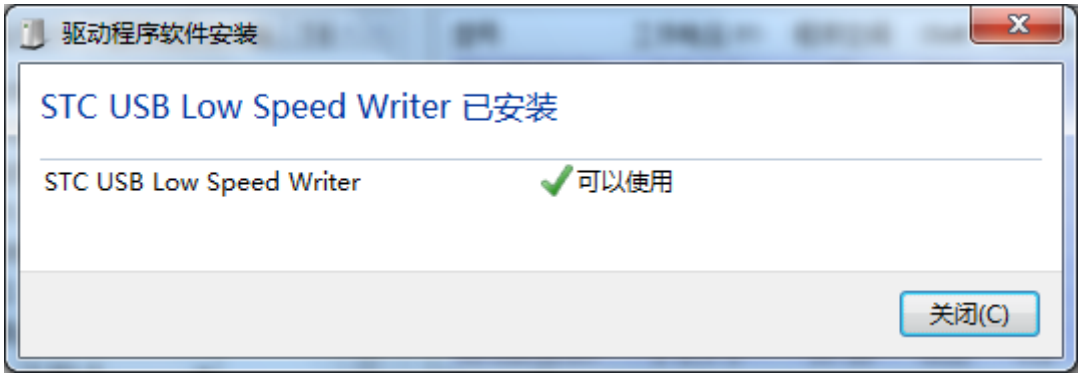


Windows 7（32 位）安装方法

打开 V6.79 版（或者更新的版本）的 STC-ISP 下载软件，下载软件会自动将驱动文件复制到相关的系统目录

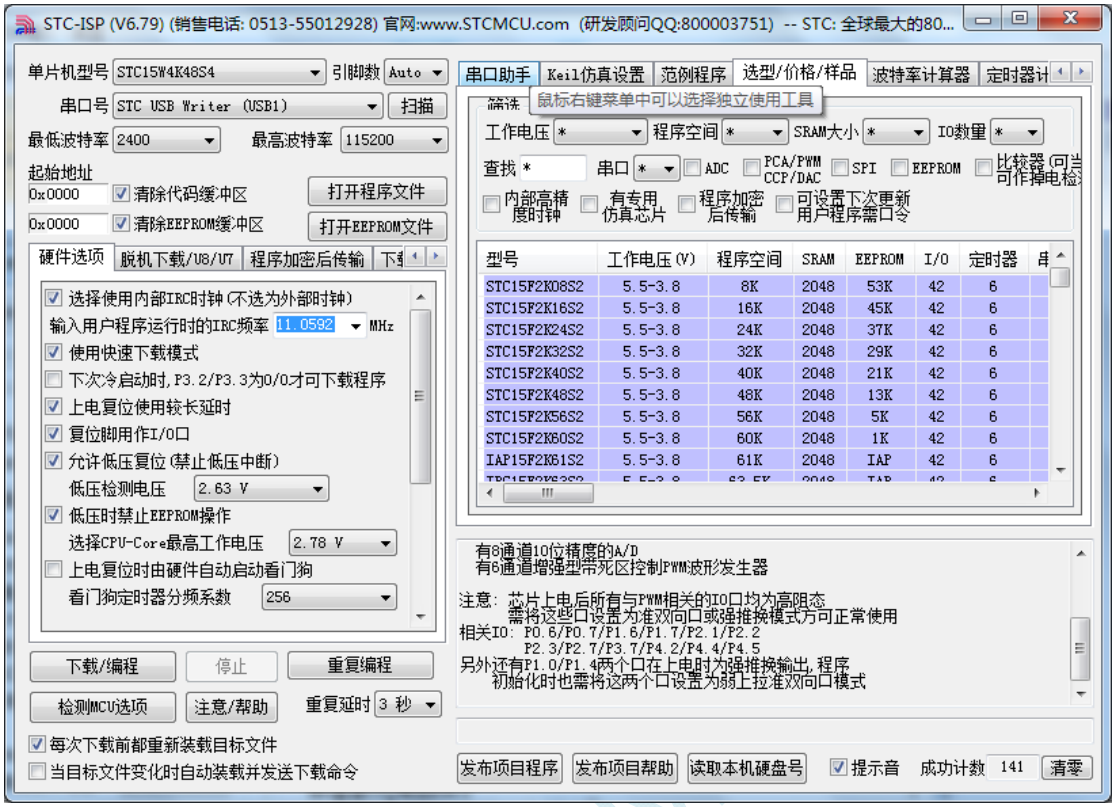


插入 USB 设备，系统找到设备后会自动安装驱动。安装完成后会有如下的提示框。



STC MCU

此时，之前打开的 STC-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)”，如下图：

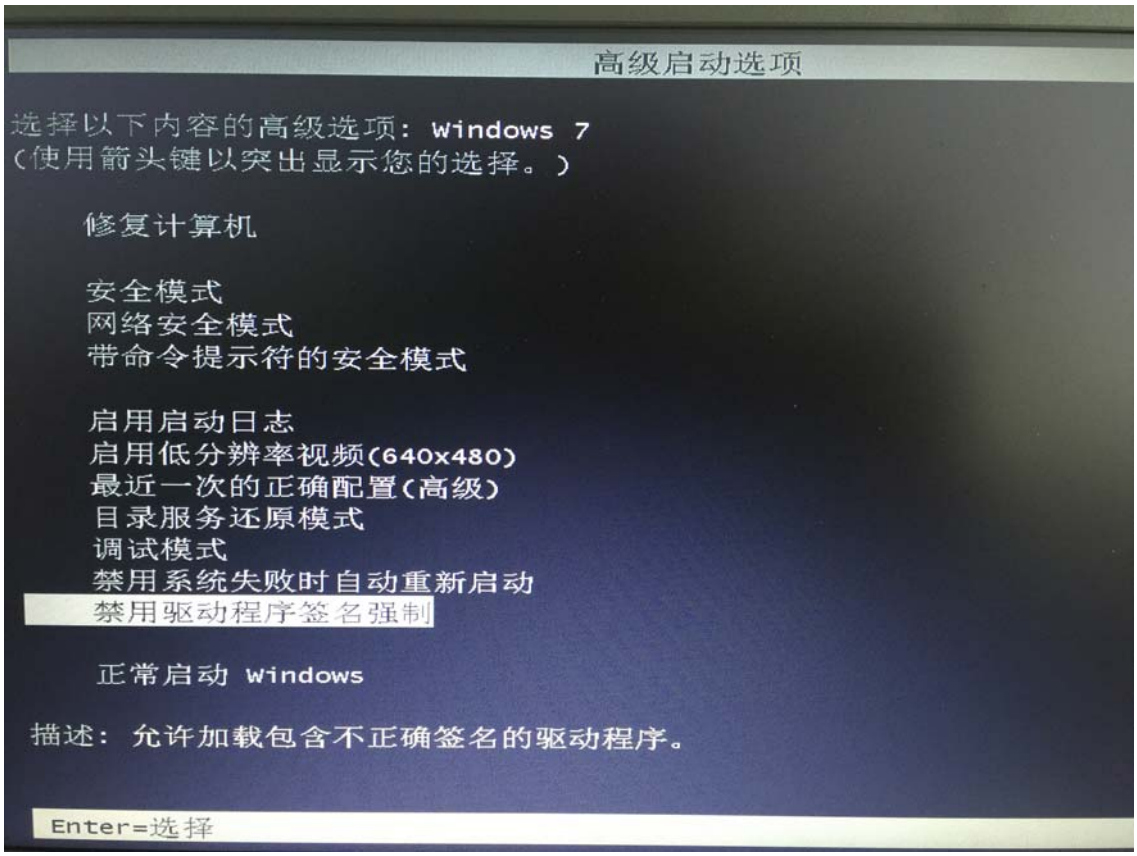


注：若 Windows 7 下，系统并没有自动安装驱动，则驱动的安装方法请参考 **Windows 8（32 位）的安装方法**

Windows 7（64 位）安装方法

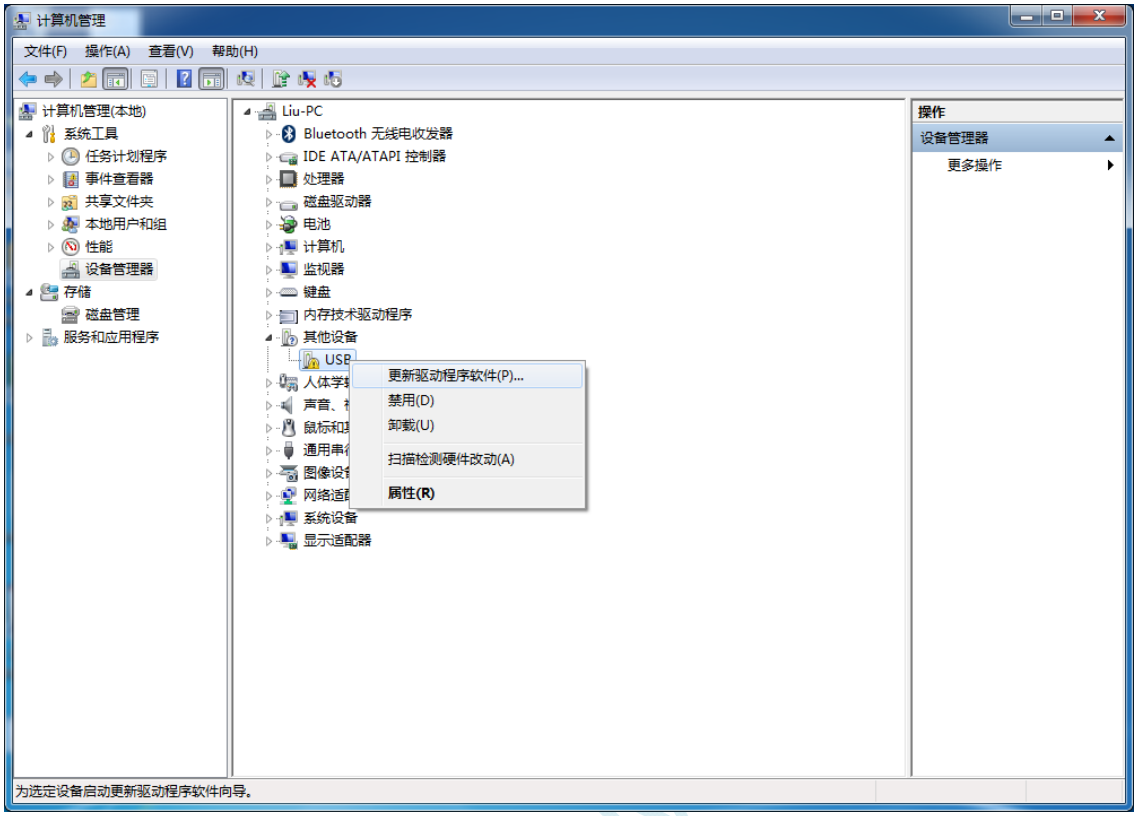
由于 Windows7 64 位操作系统在默认状态下，对于没有数字签名的驱动程序是不能安装成功的。所以在安装 STC-USB 驱动前，需要按照如下步骤，暂时跳过数字签名，即可顺利安装成功。

首先重启电脑，并一直按住 F8，直到出现下面启动画面

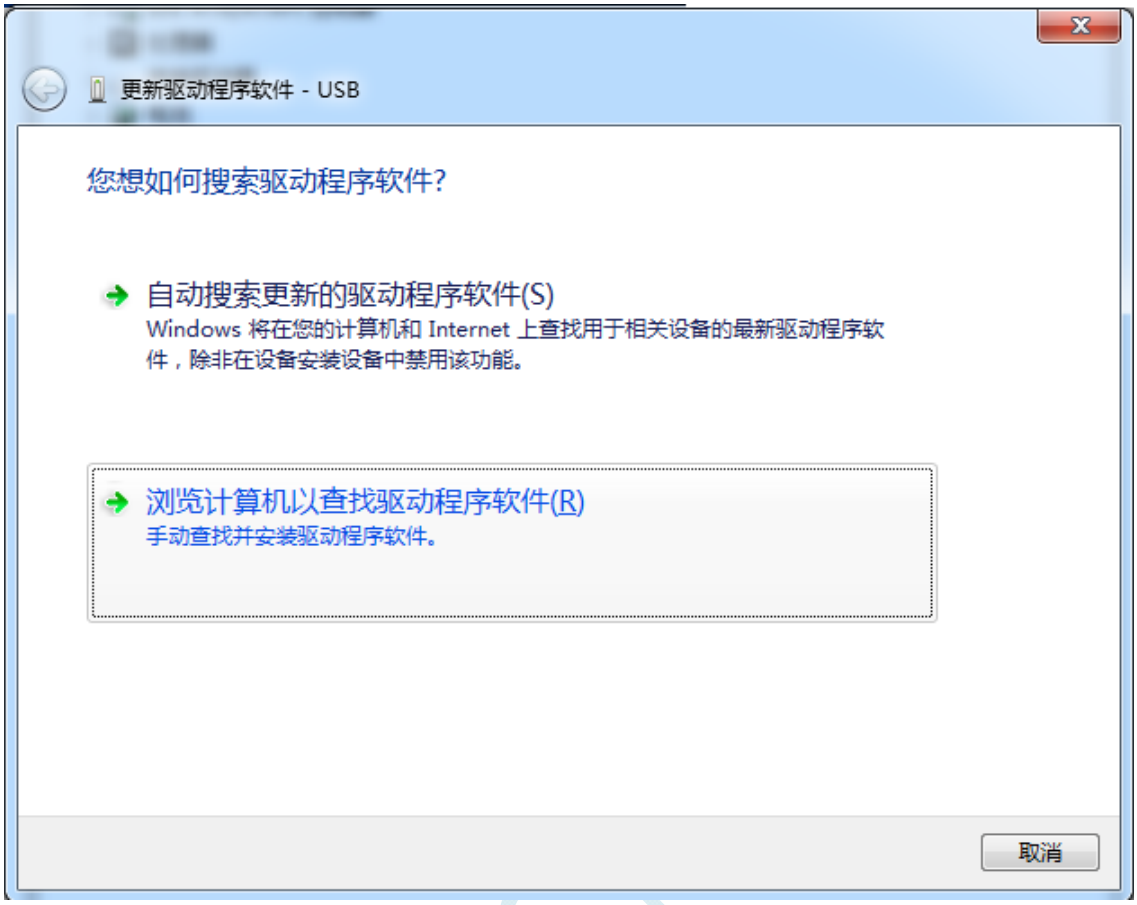


选择“禁用驱动程序签名强制”。启动后即可暂时关闭数字签名验证功能

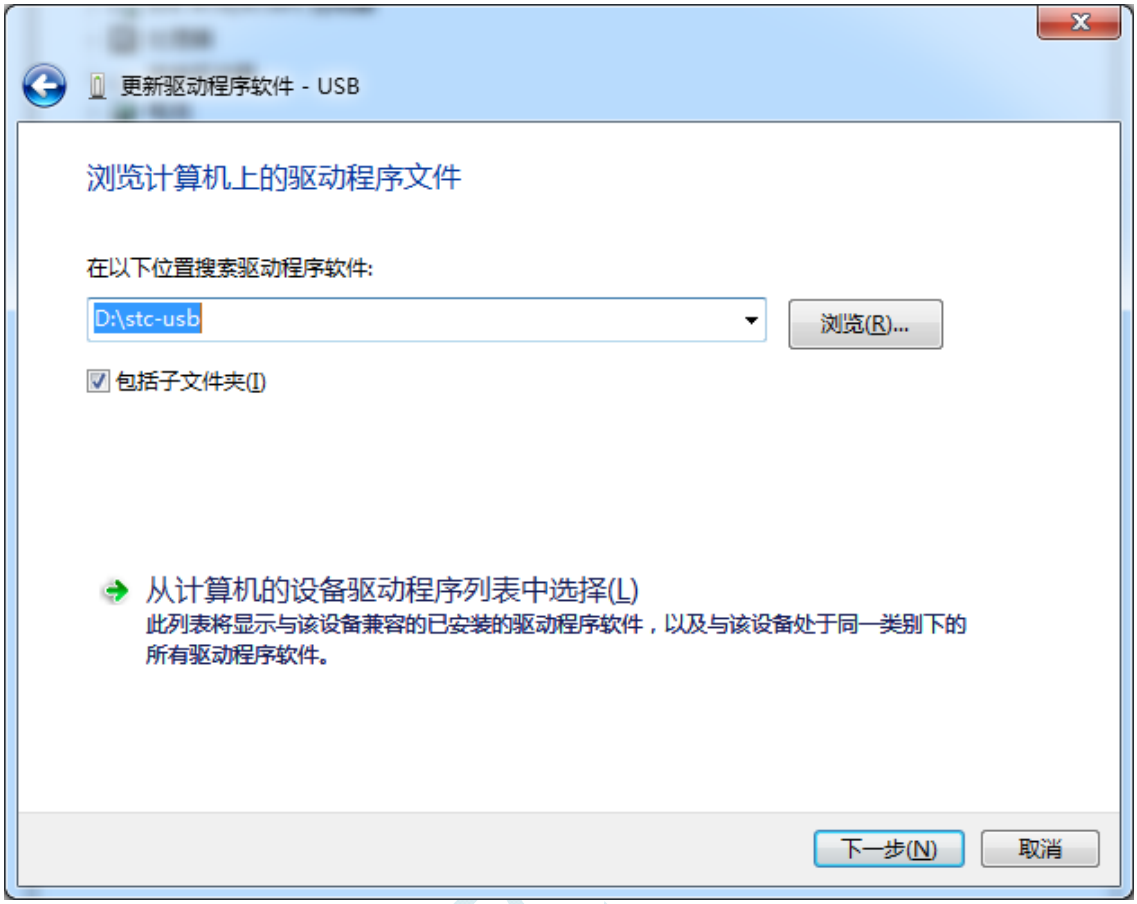
插入 USB 设备，并打开“设备管理器”。找到设备列表中带黄色感叹号的 USB 设备，在设备的右键菜单中，选择“更新驱动程序软件”



在下面的的对话框中选择“浏览计算机以查找驱动程序软件”



单击下面对话框中的“浏览”按钮，找到之前 STC-USB 驱动程序的存放目录（例如：之前的示例目录为 “ D:\STC-USB ”， 用 户 将 路 径 定 位 到 实 际 的 解 压 目 录 ）

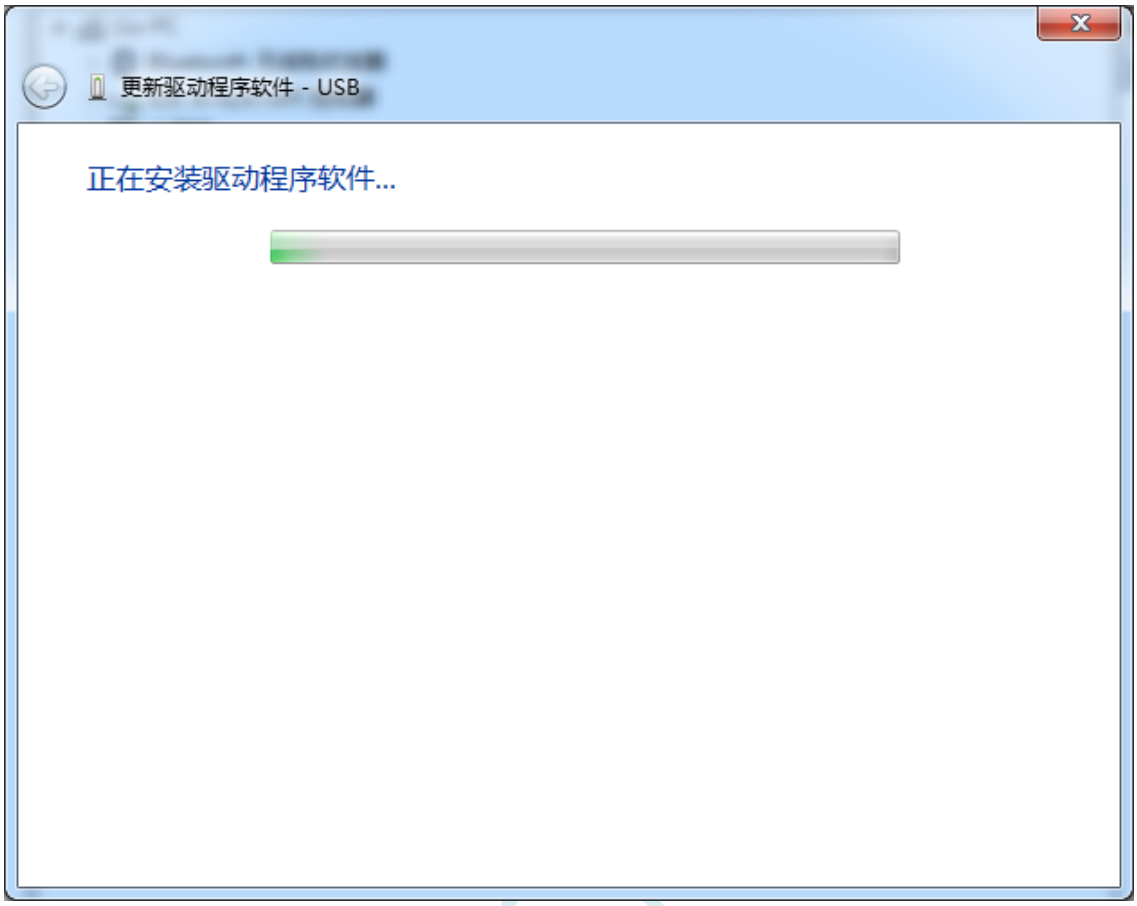


驱动程序开始安装时，会弹出如下对话框，选择“始终安装此驱动程序软件”

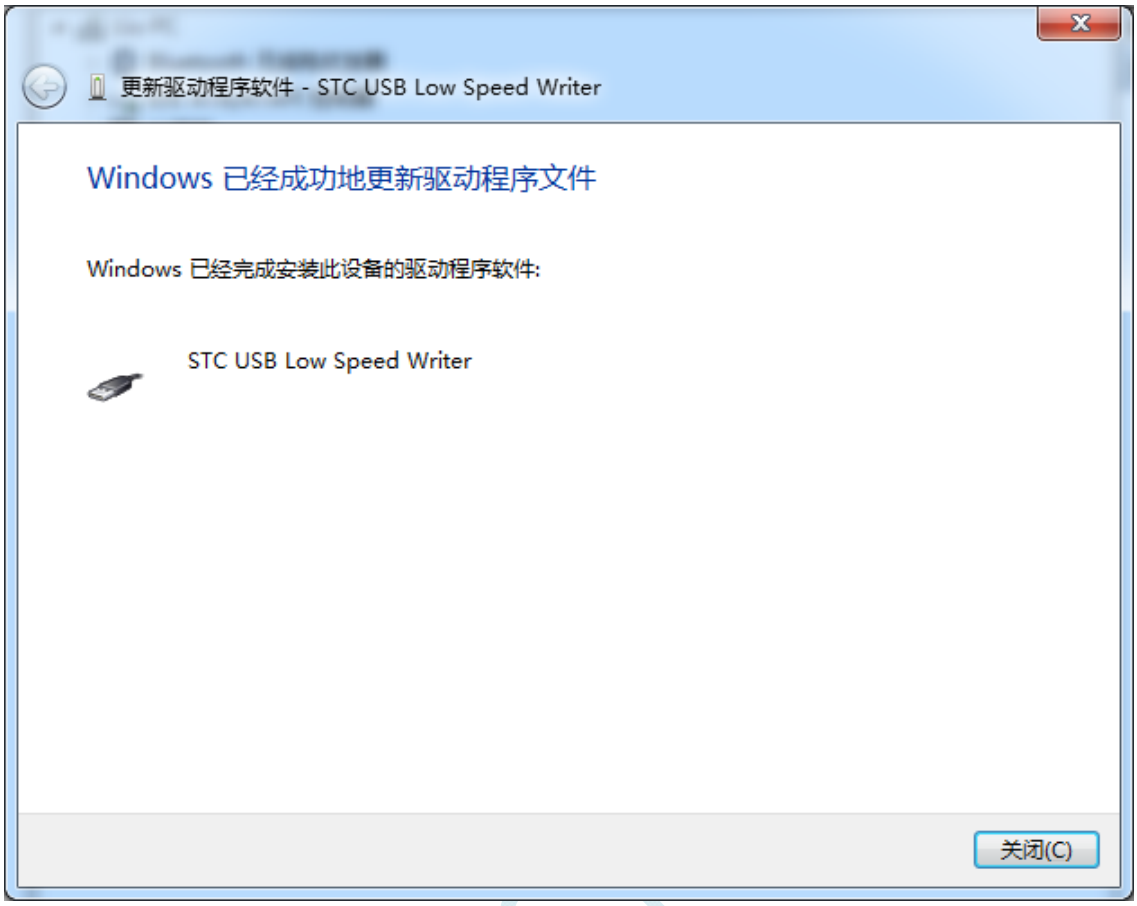


STC MCU

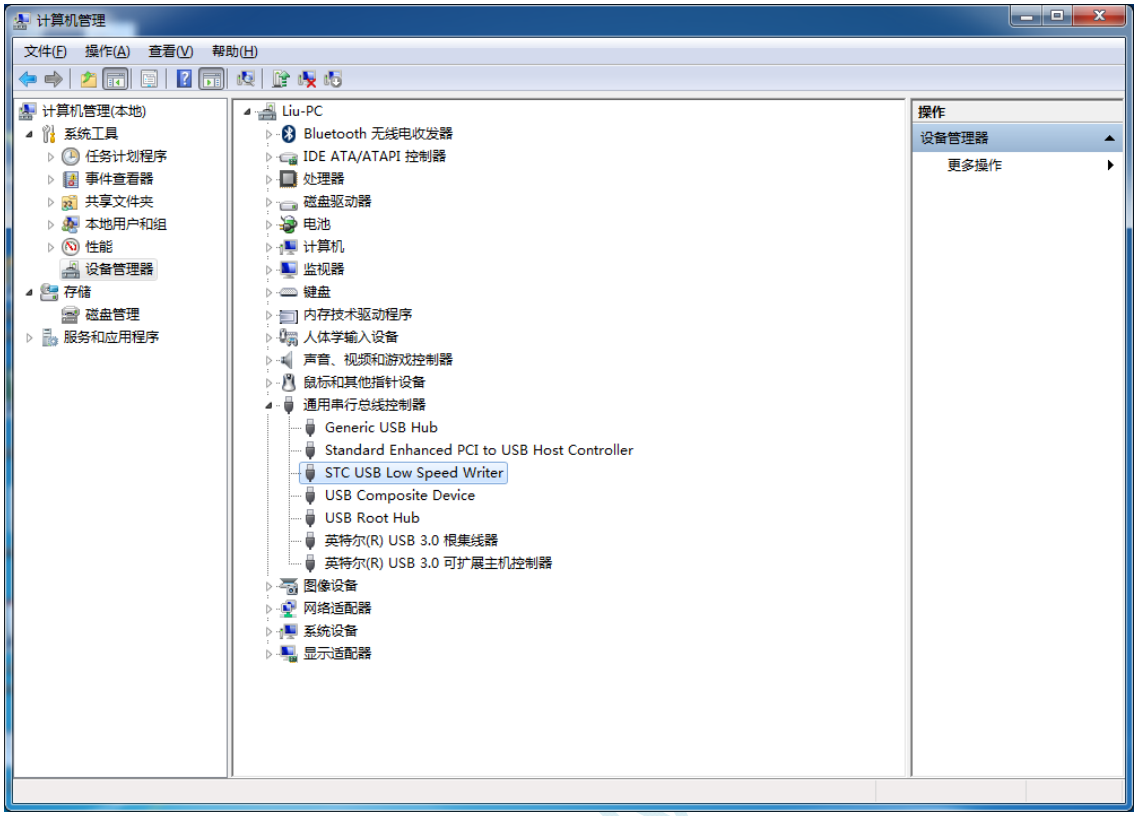
接下来，系统会自动安装驱动，如下图



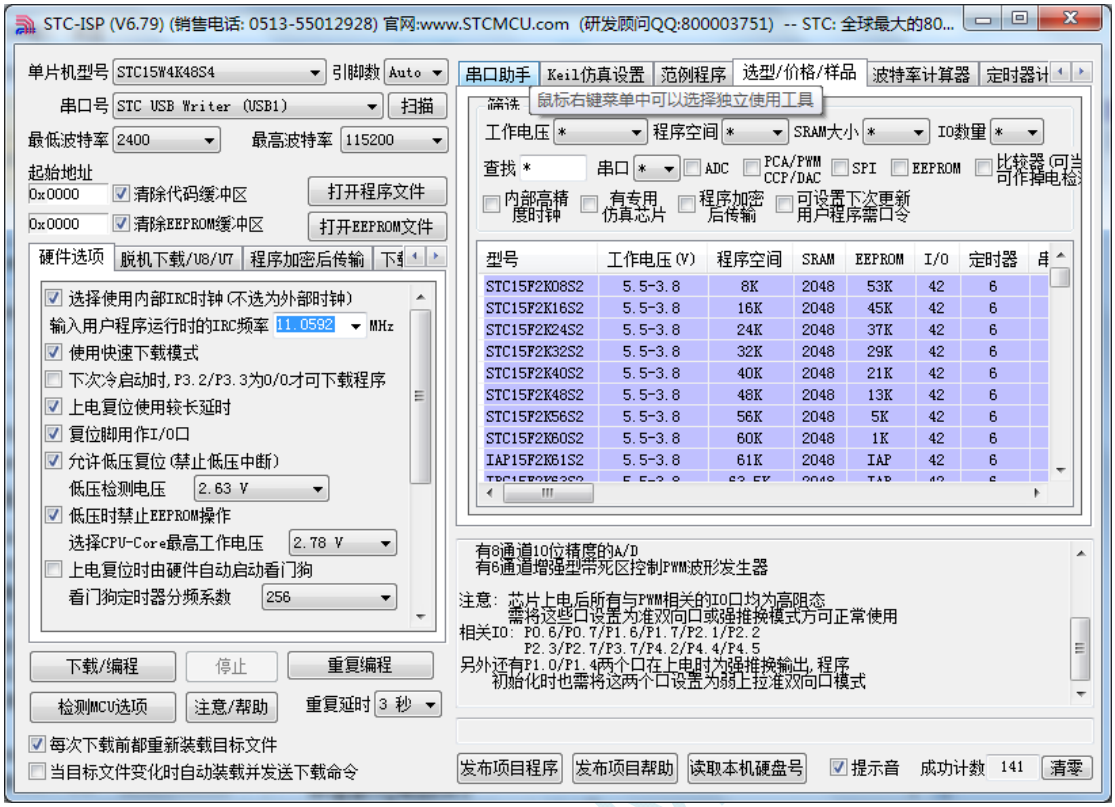
出现下面的的对话框表示驱动安装完成



此时在设备管理器中，之前带有黄色感叹号的设备，此时会显示为“STC USB Low Speed Writer”的设备名

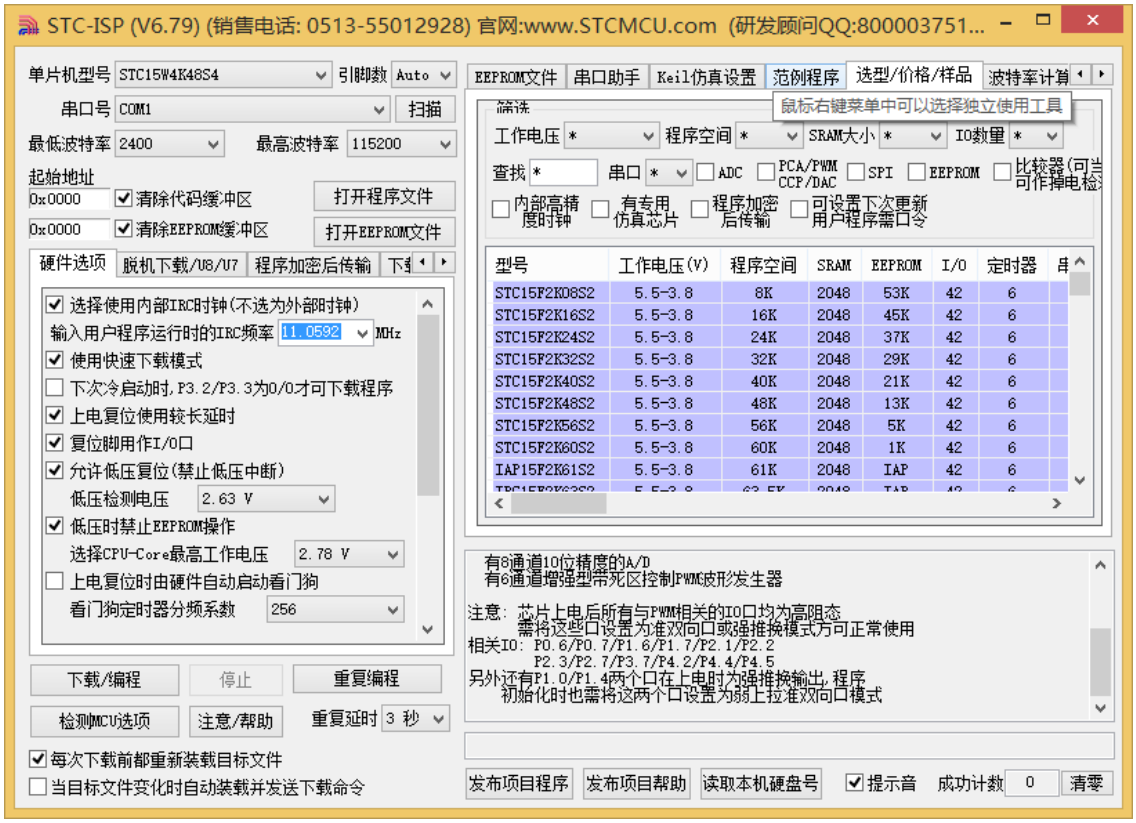


在之前打开的 STC-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)”，如下图：

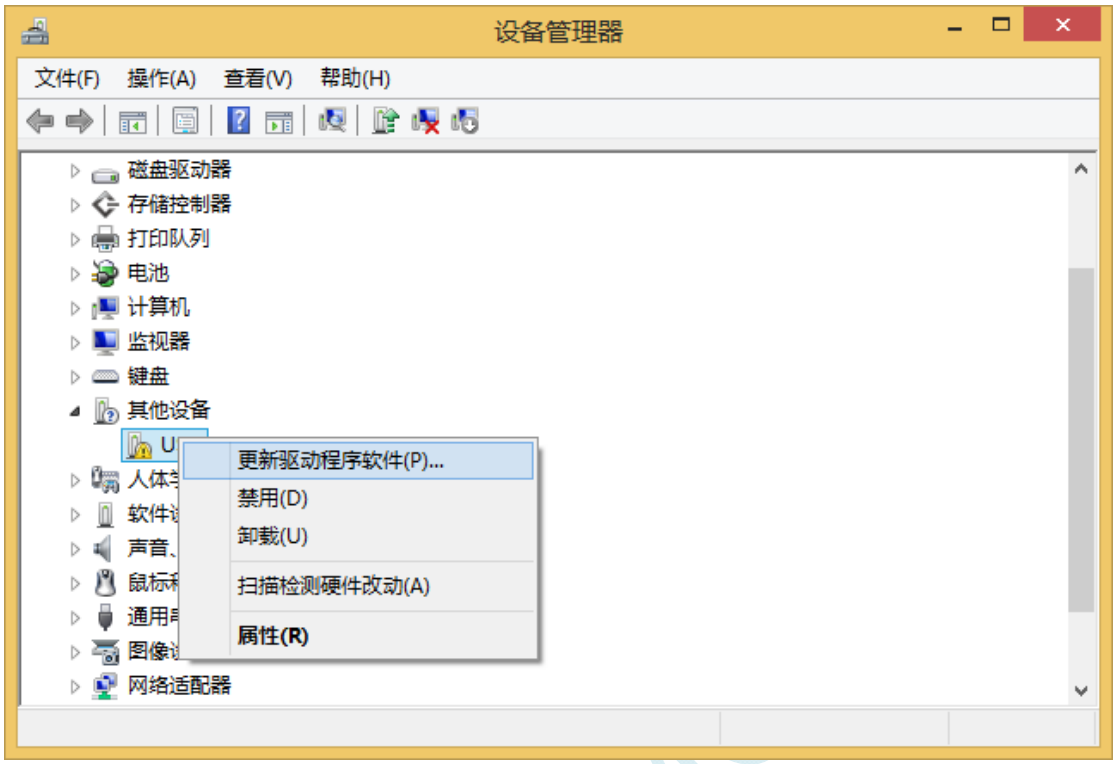


Windows 8（32 位）安装方法

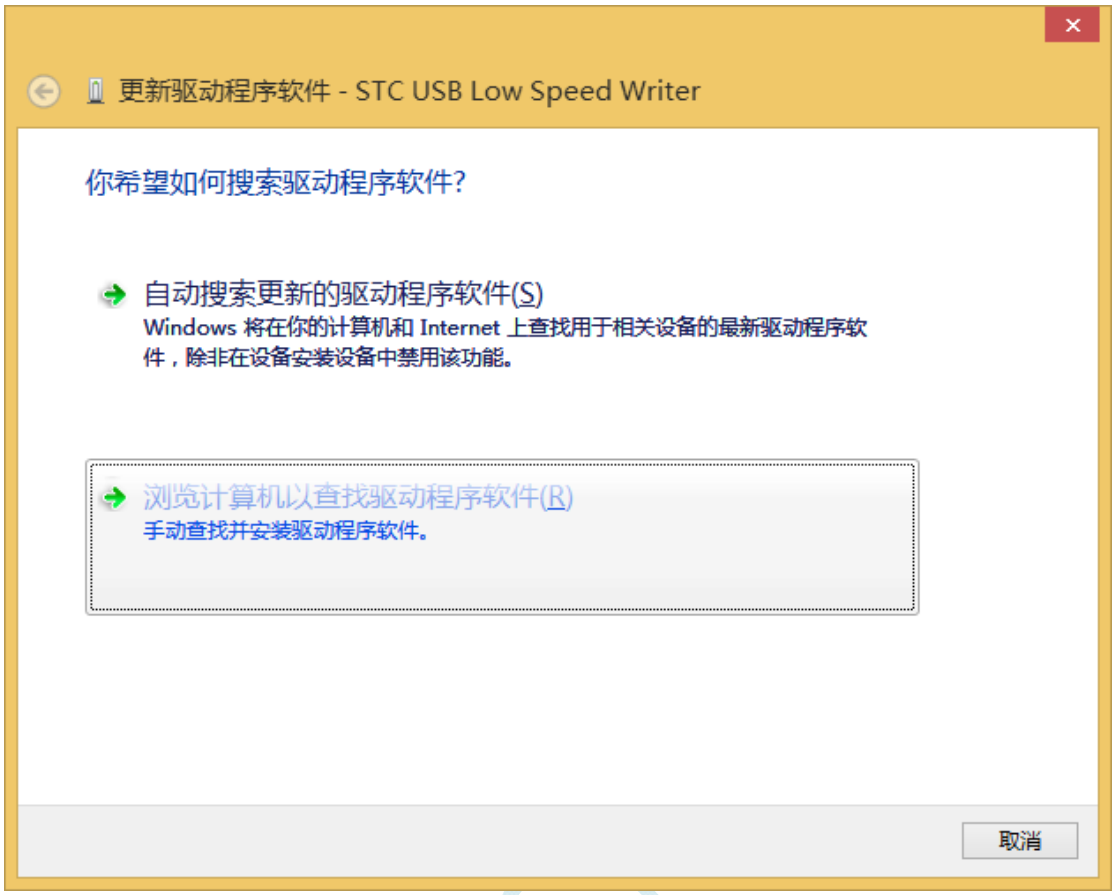
打开 V6.79 版（或者更新的版本）的 STC-ISP 下载软件（由于权限的原因，在 Windows 8 中下载软件不会将驱动文件复制到相关的系统目录，需要用户手动安装。首先从 STC 官方网站下载“stc-isp-15xx-v6.79.zip”（或更新版本），下载后解压到本地磁盘，则 STC-USB 的驱动文件也会被解压到当前解压目录中的“STC-USB Driver”中（例如将下载的压缩文件“stc-isp-15xx-v6.79.zip”解压到“F:\”，则 STC-USB 驱动程序在“F:\STC-USB Driver”目录中））



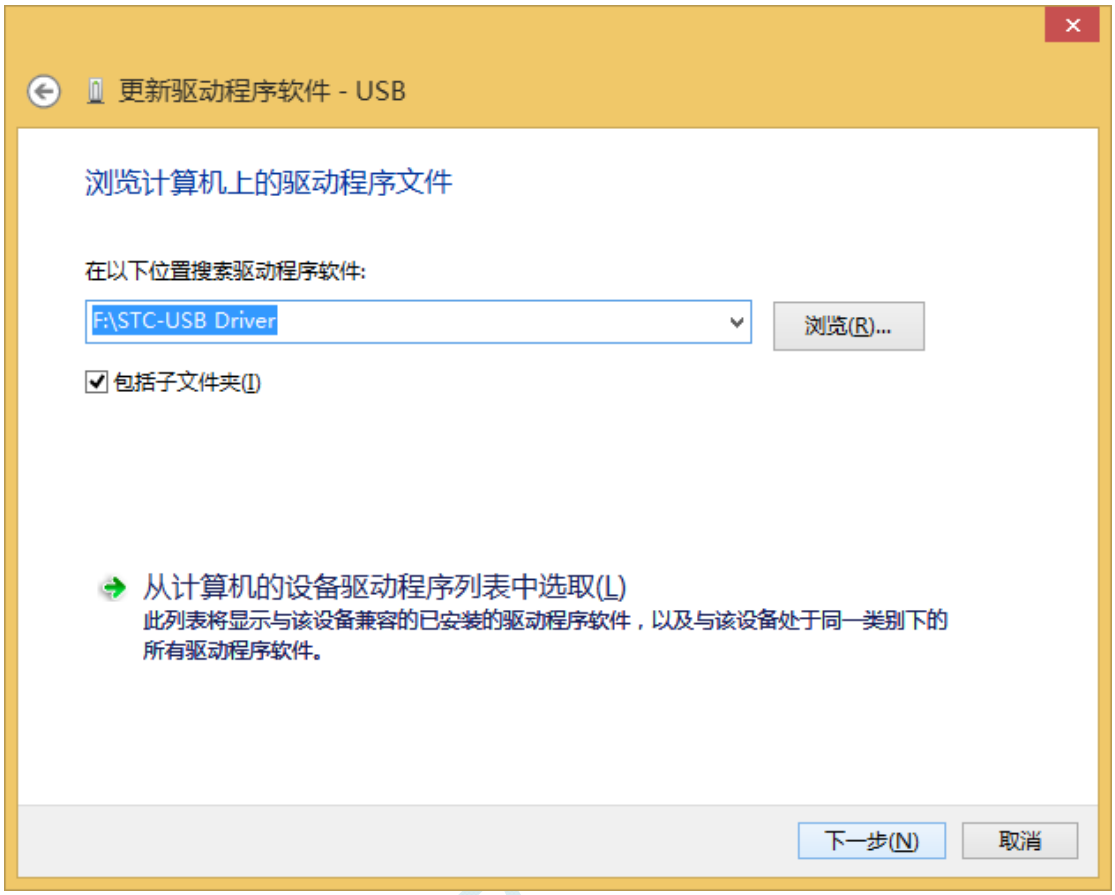
插入 USB 设备，并打开“设备管理器”。找到设备列表中带黄色感叹号的 USB 设备，在设备的右键菜单中，选择“更新驱动程序软件”



在下面的的对话框中选择“浏览计算机以查找驱动程序软件”



单击下面对话框中的“浏览”按钮，找到之前 STC-USB 驱动程序的存放目录（例如：之前的示例目录为“F:\STC-USB Driver”，用户将路径定位到实际的解压目录）

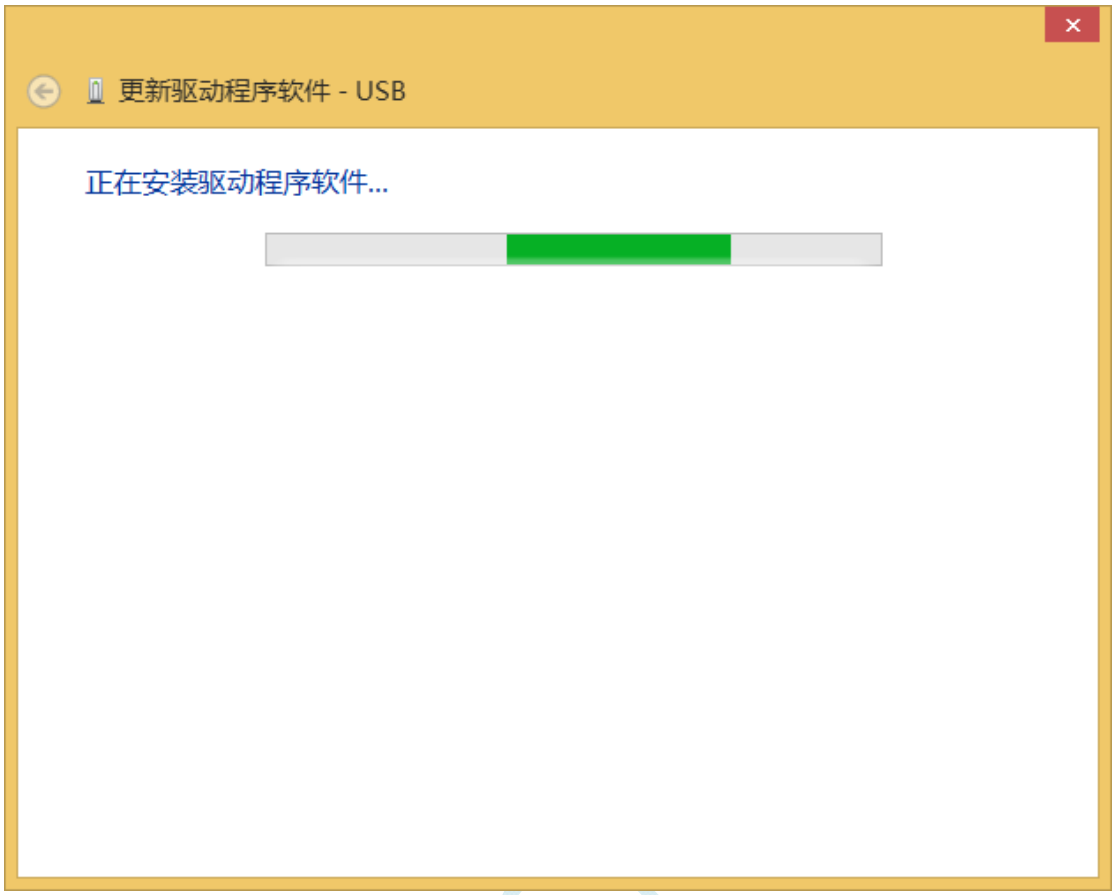


驱动程序开始安装时，会弹出如下对话框，选择“始终安装此驱动程序软件”

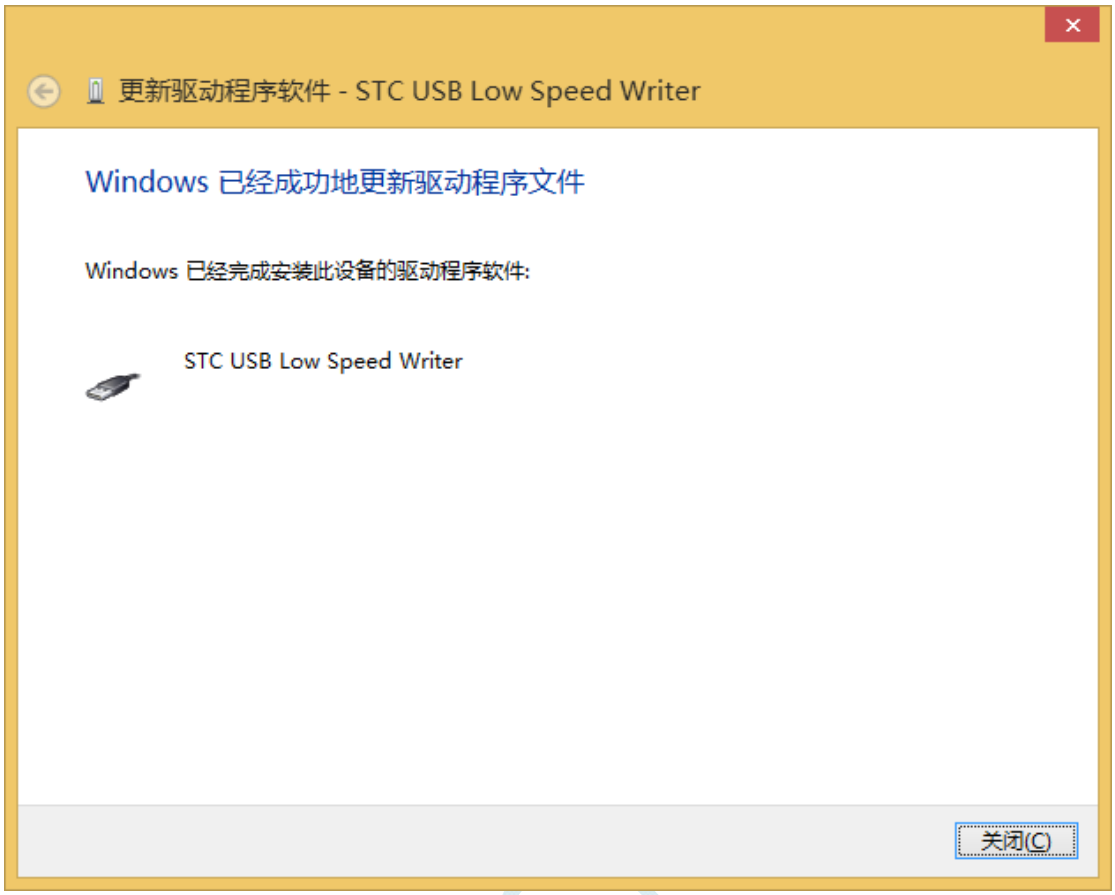


STC MCU

接下来，系统会自动安装驱动，如下图



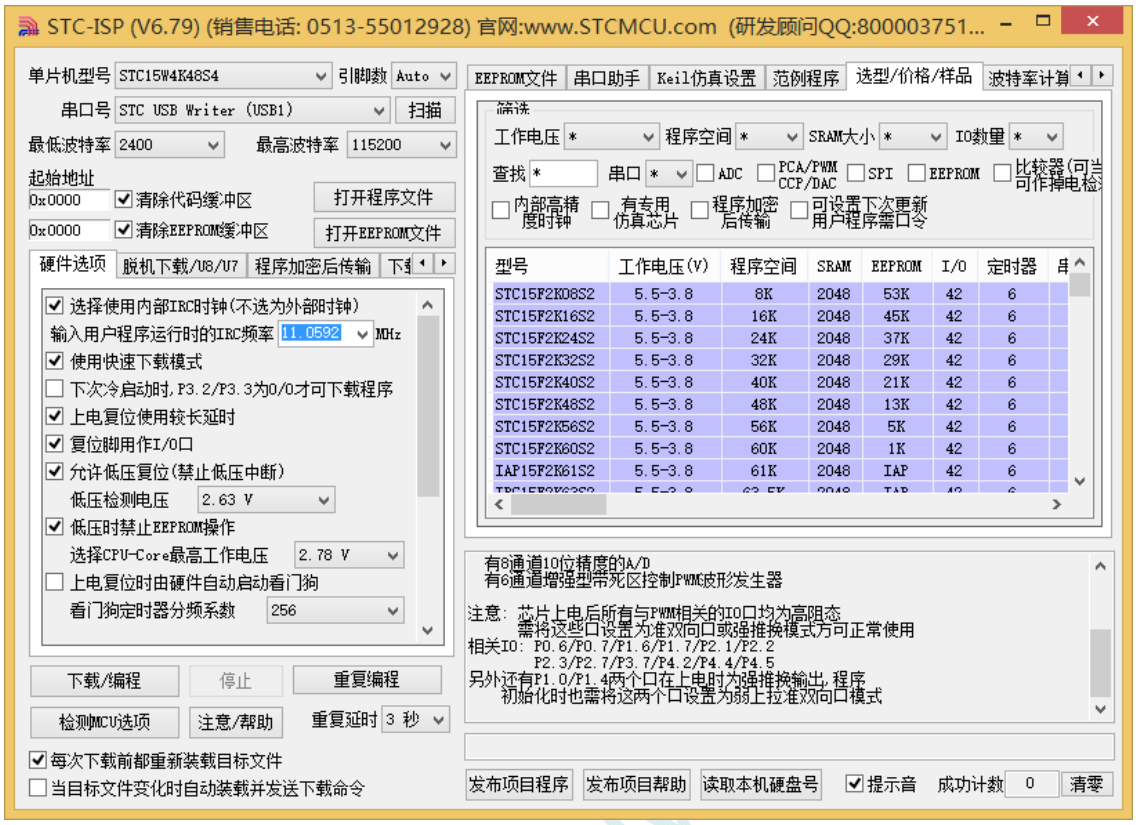
出现下面的的对话框表示驱动安装完成



此时在设备管理器中，之前带有黄色感叹号的设备，此时会显示为“STC USB Low Speed Writer”的设备名



在之前打开的 STC-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)”，如下图：



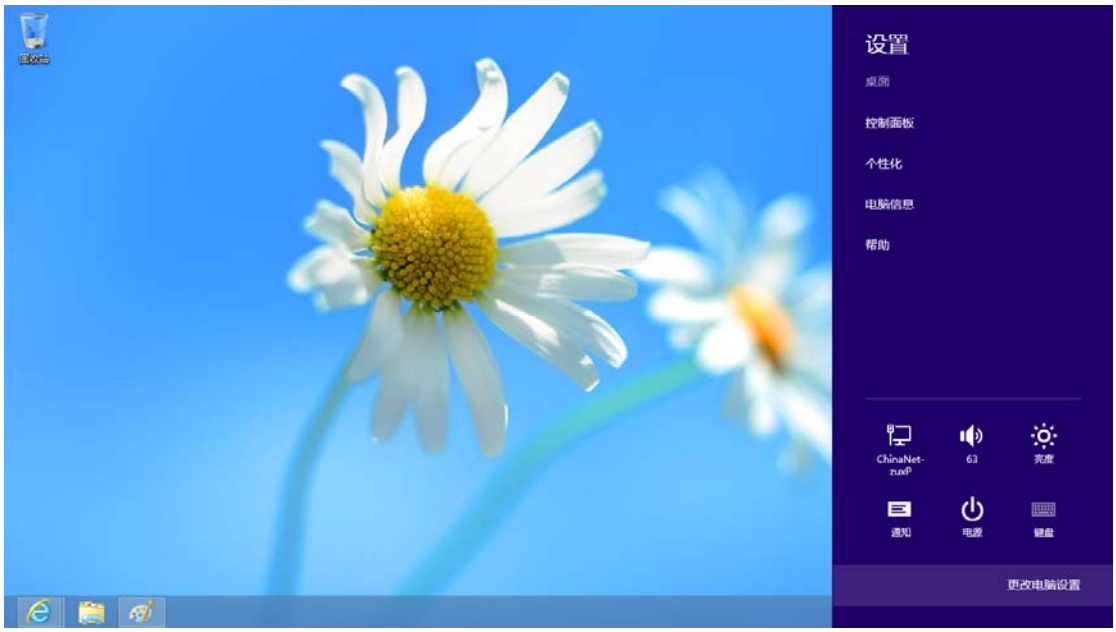
Windows 8（64 位）安装方法

由于 Windows8 64 位操作系统在默认状态下，对于没有数字签名的驱动程序是不能安装成功的。所以在安装 STC-USB 驱动前，需要按照如下步骤，暂时跳过数字签名，即可顺利安装成功。

首先将鼠标移动到屏幕的右下角，选择其中的“设置”按钮



然后在设置界面中选择“更改电脑设置”项



STC MCU

在电脑设置中，选择“常规”属性页中“高级启动”项下面的“立即启动”按钮



STC MCU

在下面的界面中，选择“疑难解答”项



然后选择“疑难解答”中的“高级选项”



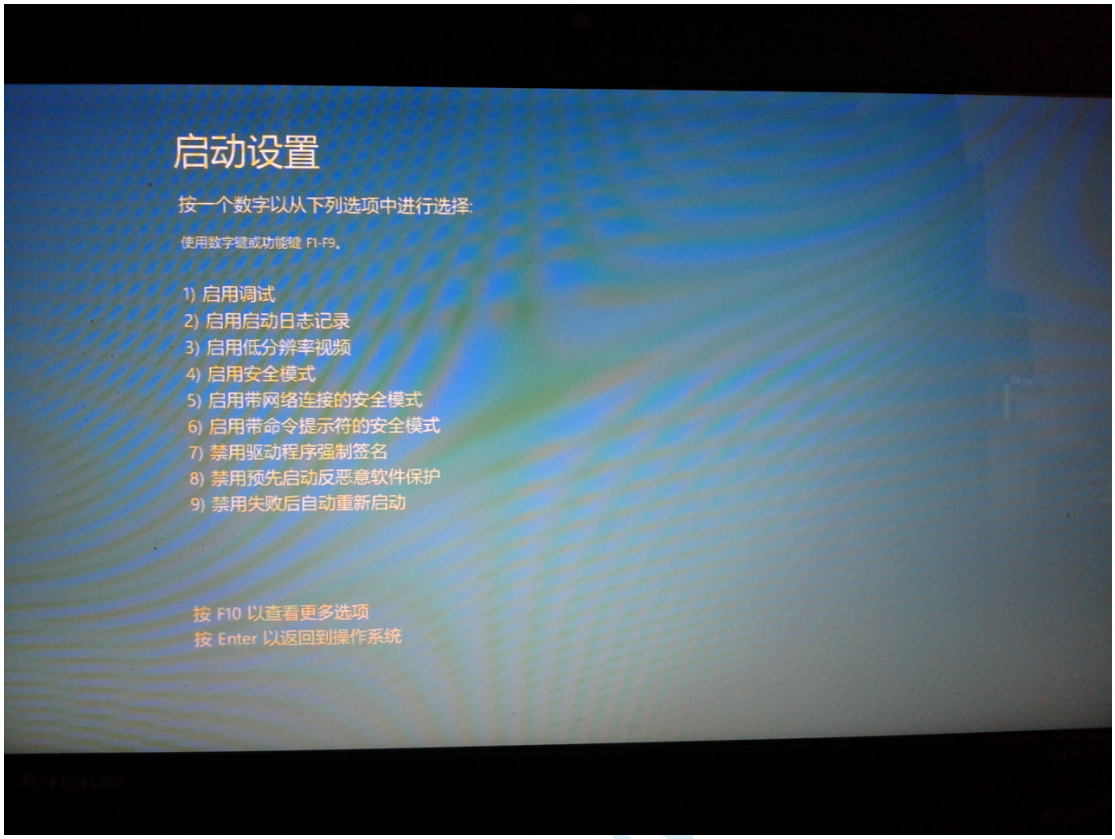
在下面的“高级选项”界面中，选择“启动设置”



在下面的“启动设置”界面中，单击“重启”按钮对电脑进行重新启动



在电脑重新启动后会自动进入如下图所示的“启动设置”界面，按数字键“7”或者按功能键“F7”选择“禁用驱动程序强制签名”进行启动



启动到 Windows 8 后，按照 **Windows 8（32 位）的安装方法**即可完成驱动的安装

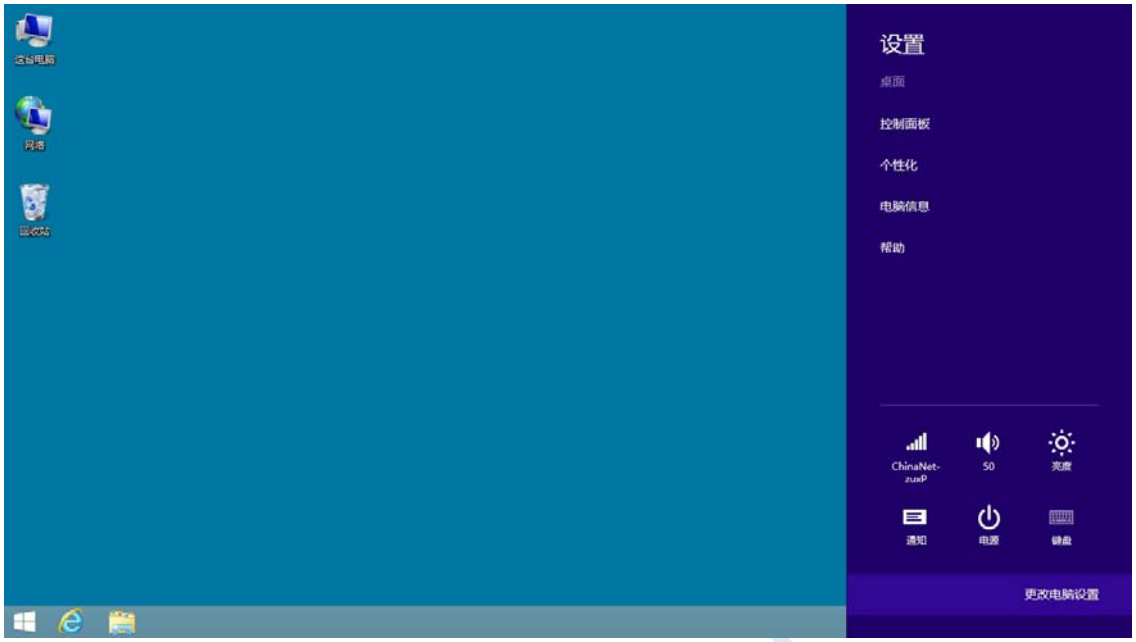
Windows 8.1（64 位）安装方法

Windows 8.1 与 Windows 8 进入高级启动菜单的方法不一样,在此专门进行说明。

首先将鼠标移动到屏幕的右下角，选择其中的“设置”按钮

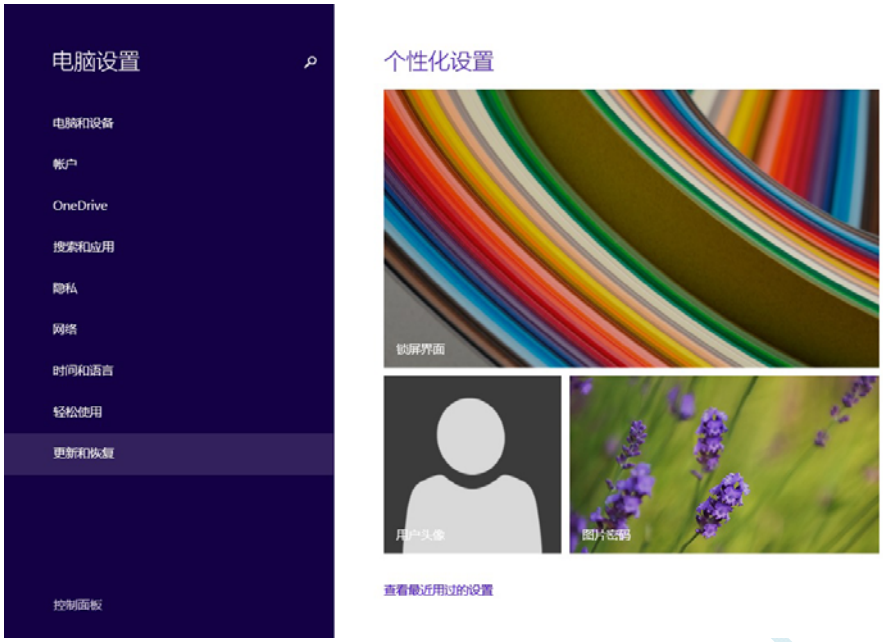


然后在设置界面中选择“更改电脑设置”项



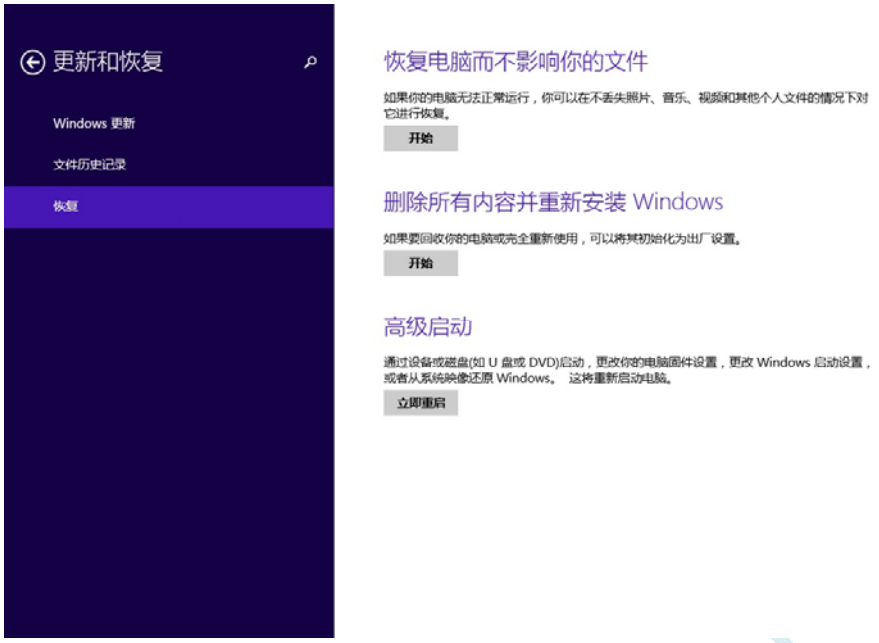
STC MCU

在电脑设置中，选择“更新和恢复”（这里与 Windows 8 不一样，Windows 8 选择的是“常规”）



STC MCU

在更新和恢复页面中选择“恢复”属性页，单击“高级启动”项下面的“立即启动”按钮



STC MCU

接下来的操作与 Window 8 的步骤相同
在下面的界面中，选择“疑难解答”项



然后选择“疑难解答”中的“高级选项”



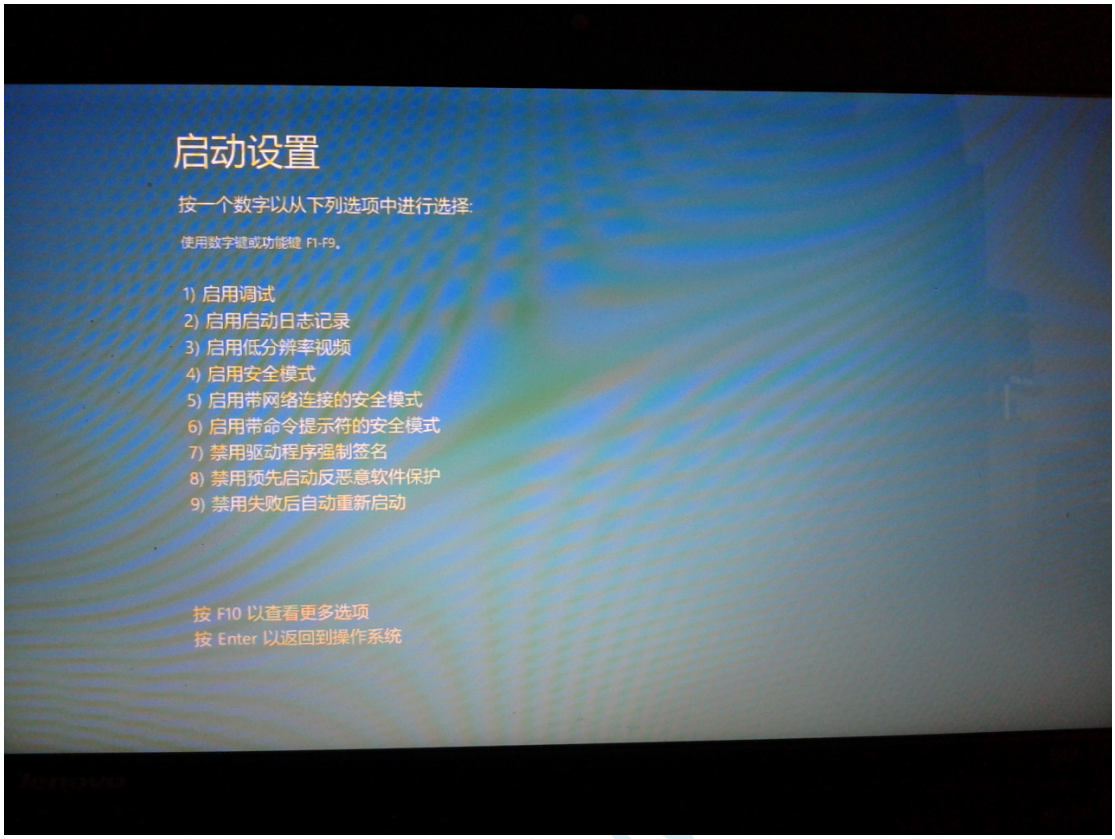
在下面的“高级选项”界面中，选择“启动设置”



在下面的“启动设置”界面中，单击“重启”按钮对电脑进行重新启动



在电脑重新启动后会自动进入如下图所示的“启动设置”界面，按数字键“7”或者按功能键“F7”选择“禁用驱动程序强制签名”进行启动



启动到 Windows 8 后，按照 **Windows 8（32 位）的安装方法**即可完成驱动的安装

Windows10（64 位）安装方法

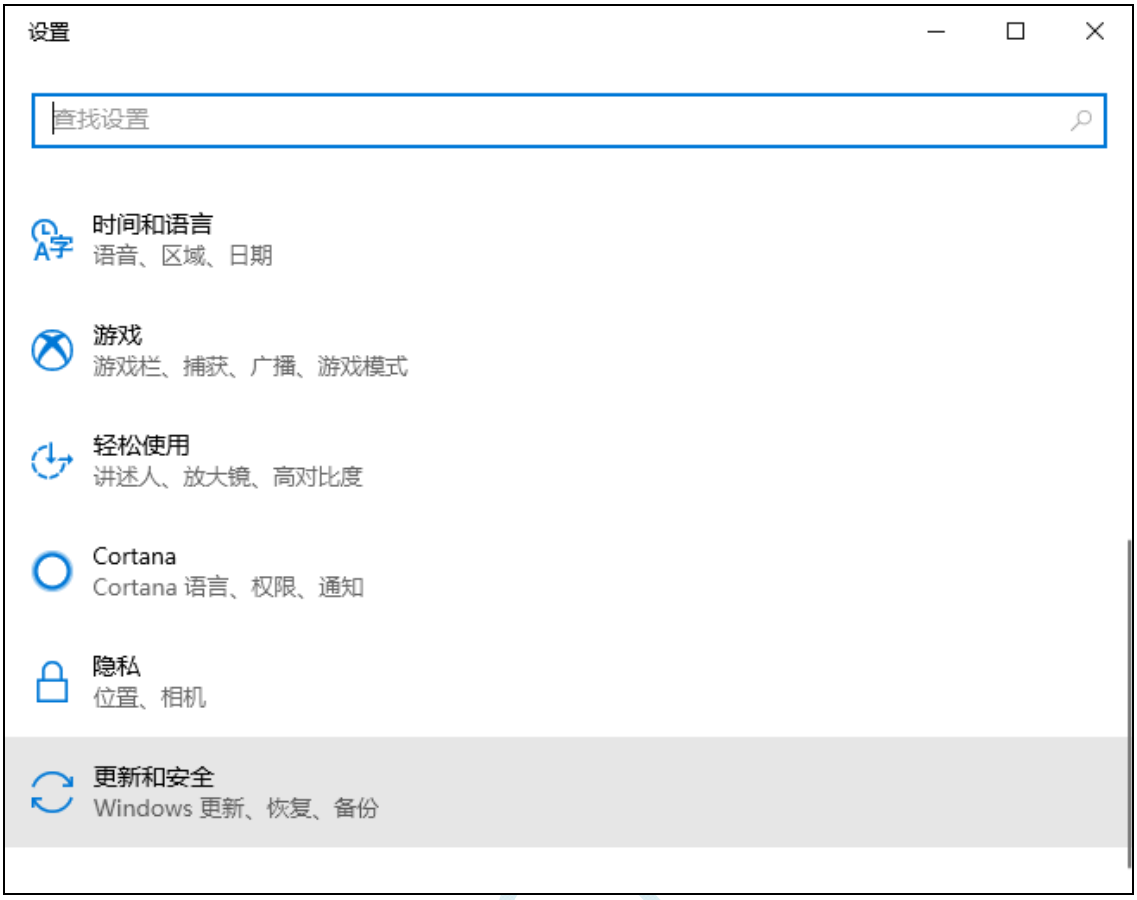
由于 Windows10 64 位操作系统在默认状态下，对于没有数字签名的驱动程序是不能安装成功的。所以在安装 STC-USB 驱动前，需要按照如下步骤，暂时跳过数字签名，即可顺利安装成功。

安装驱动前需要从 STC 官网下载的 STC-ISP 下载软件压缩包中将“STC-USB Driver”文件夹解压缩到硬盘中。将具有 USB 下载功能的芯片准备好，但先不要连接电脑

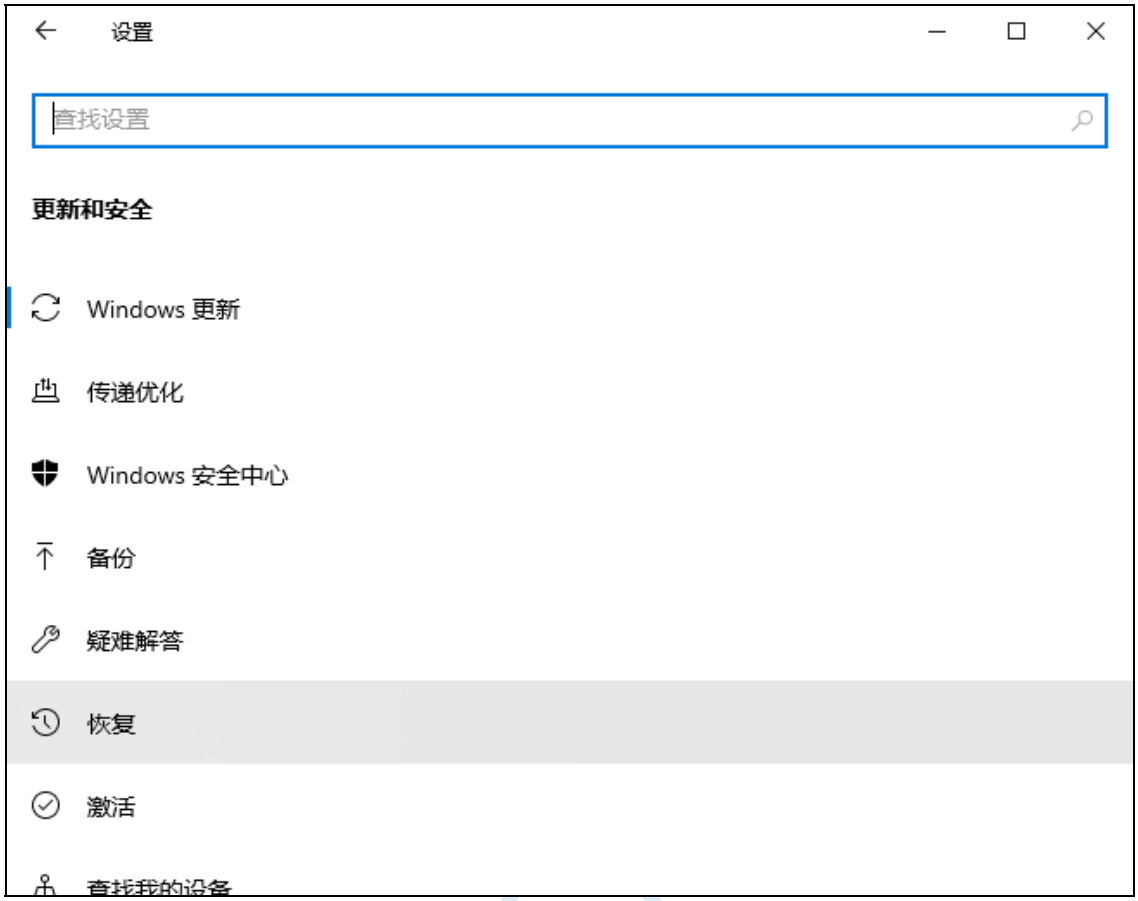
鼠标右键点击“开始”菜单，选择“设置”选项



然后在设置界面中选择“更新和安全”项



然后在设置界面中选择“恢复”项



在恢复界面中，点击“高级启动”项中的“立即重新启动”按钮



在电脑重启前，系统会先进入如下的启动菜单，选择“疑难解答”项



在疑难解答界面中选择“高级选项”



然后选择“查看更多恢复选项”



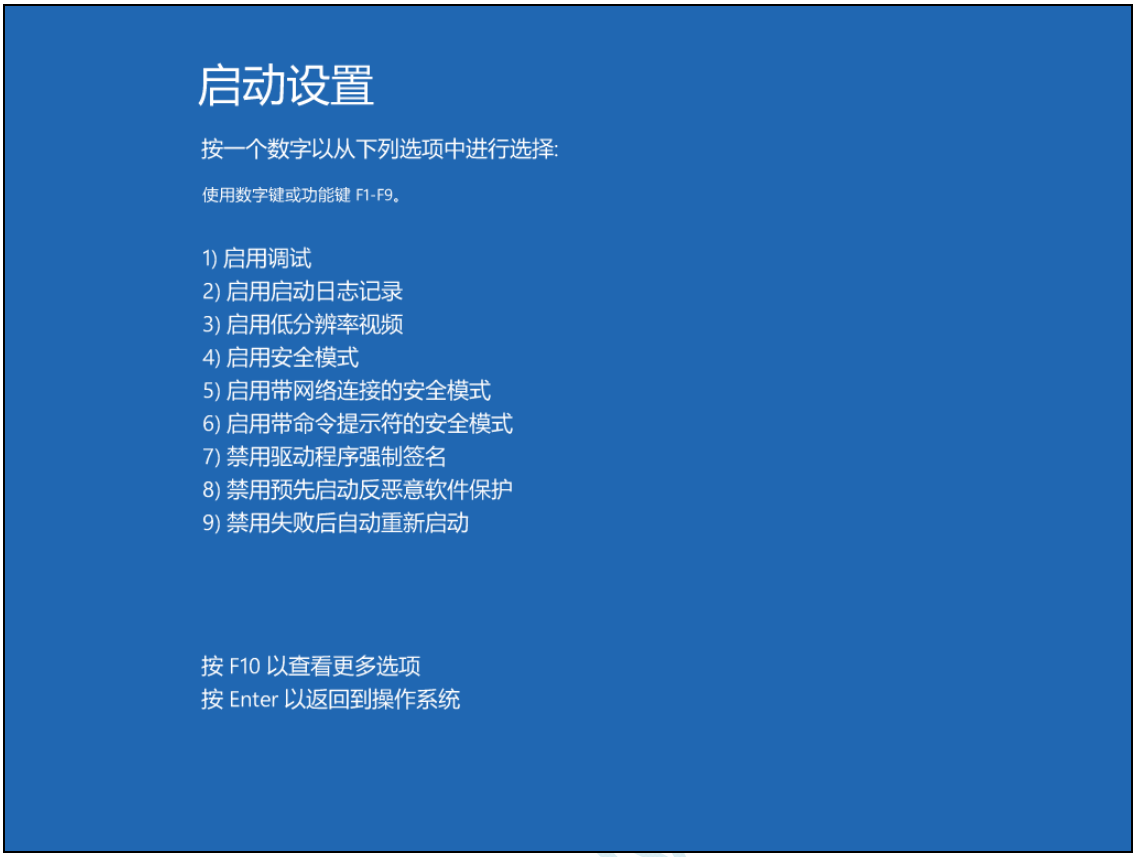
选择“启动设置”项



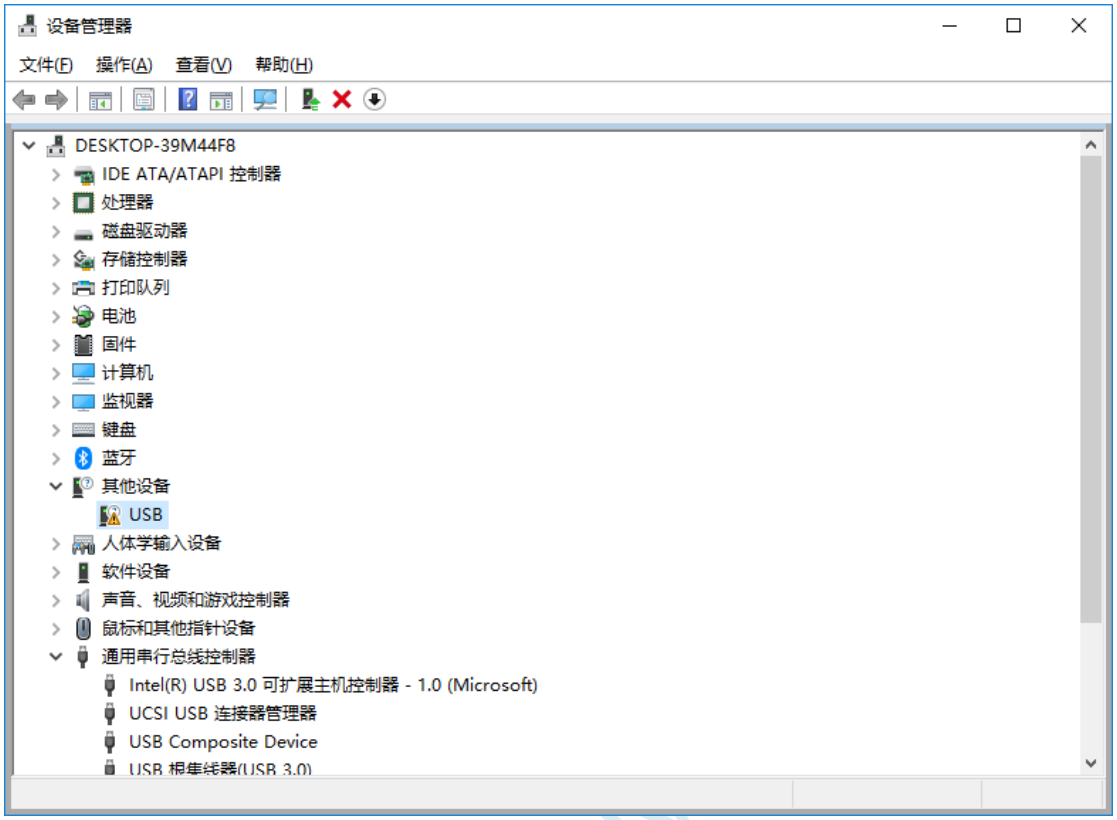
出现如下画面后，点击“重启”按钮重启电脑



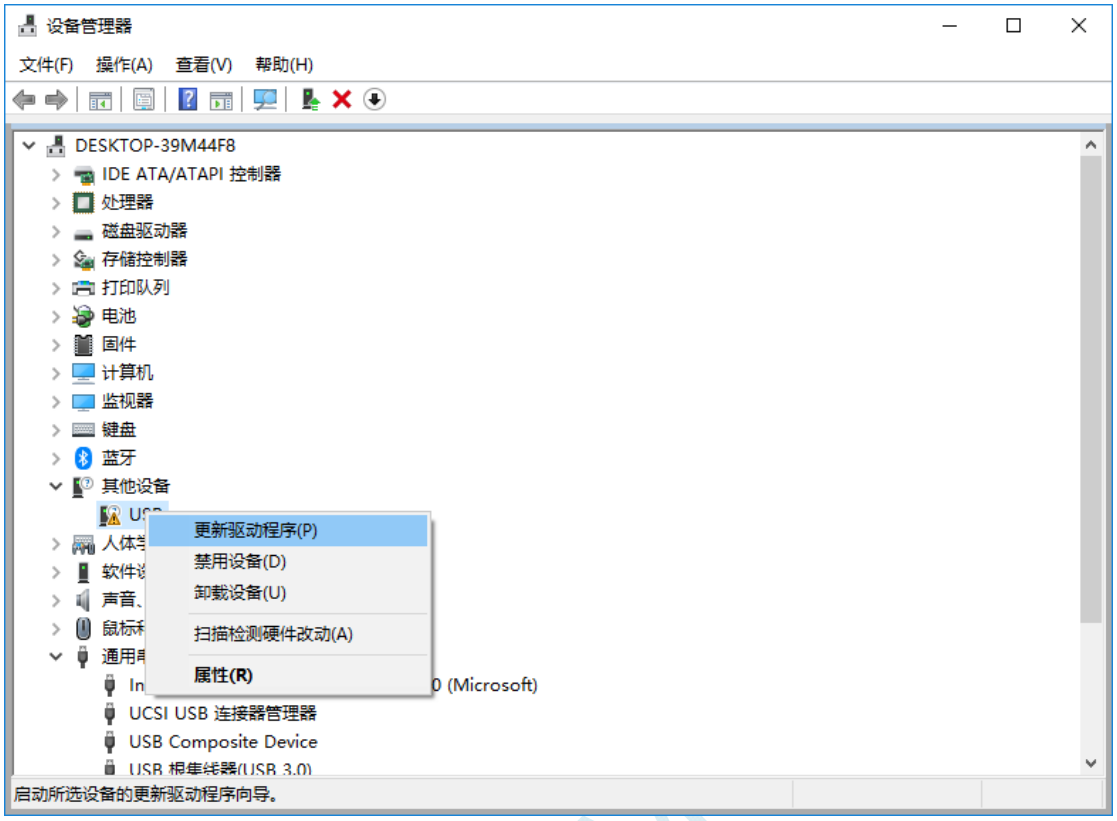
电脑重启后，会弹出“启动设置”界面，按“F7”按钮来选择“禁止驱动程序强制签名”项



电脑启动完成后，将准备好的芯片用 USB 线与电脑相连，并打开“设备管理器”，此时由于驱动还没有开始安装，所以在设备管理器中会显示为一个带感叹号的未知设备



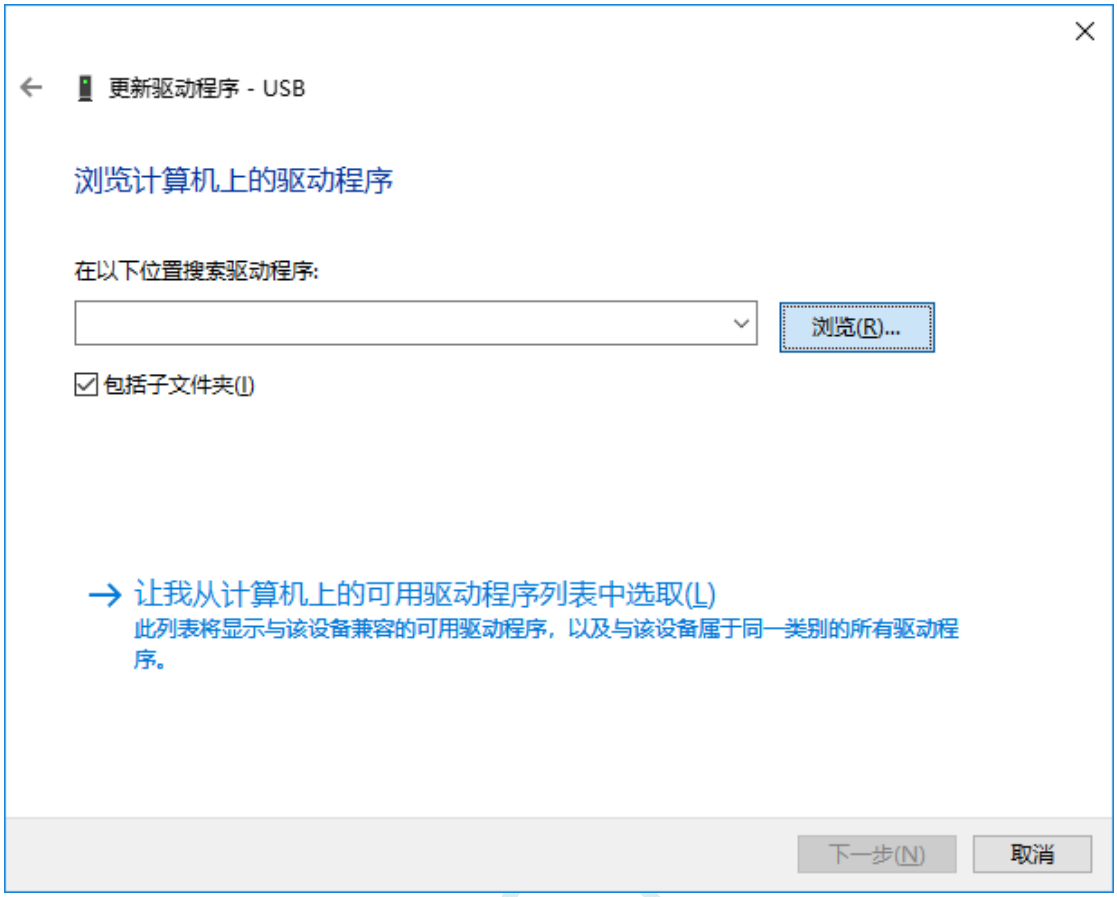
鼠标右键单击未知设备，选择右键菜单中的“更新驱动程序”



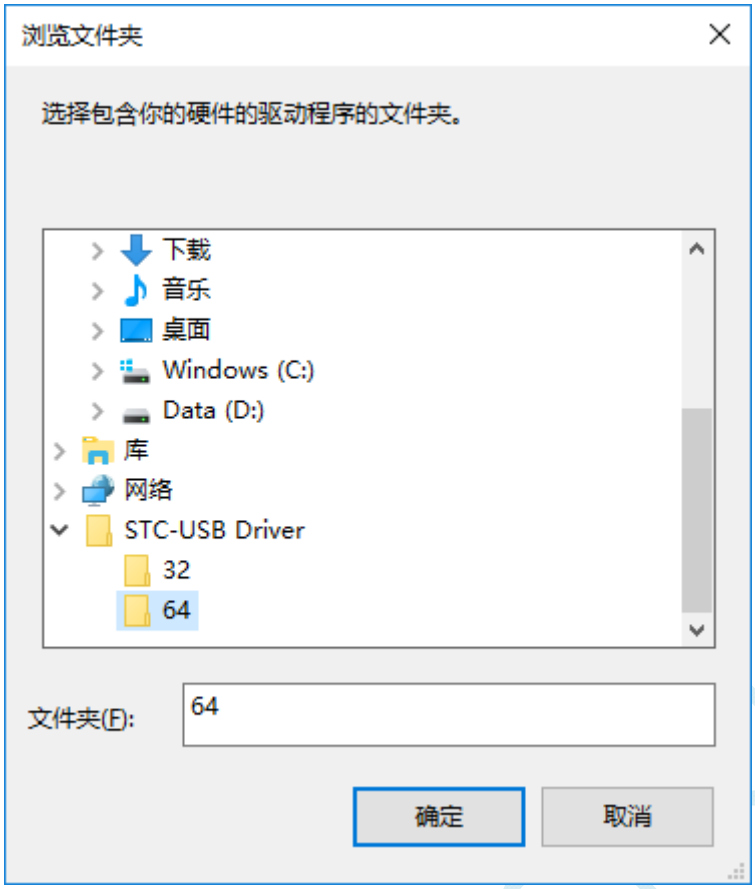
在弹出的驱动安装程序选择画面中，选择“浏览我的计算机以查找驱动程序软件”项



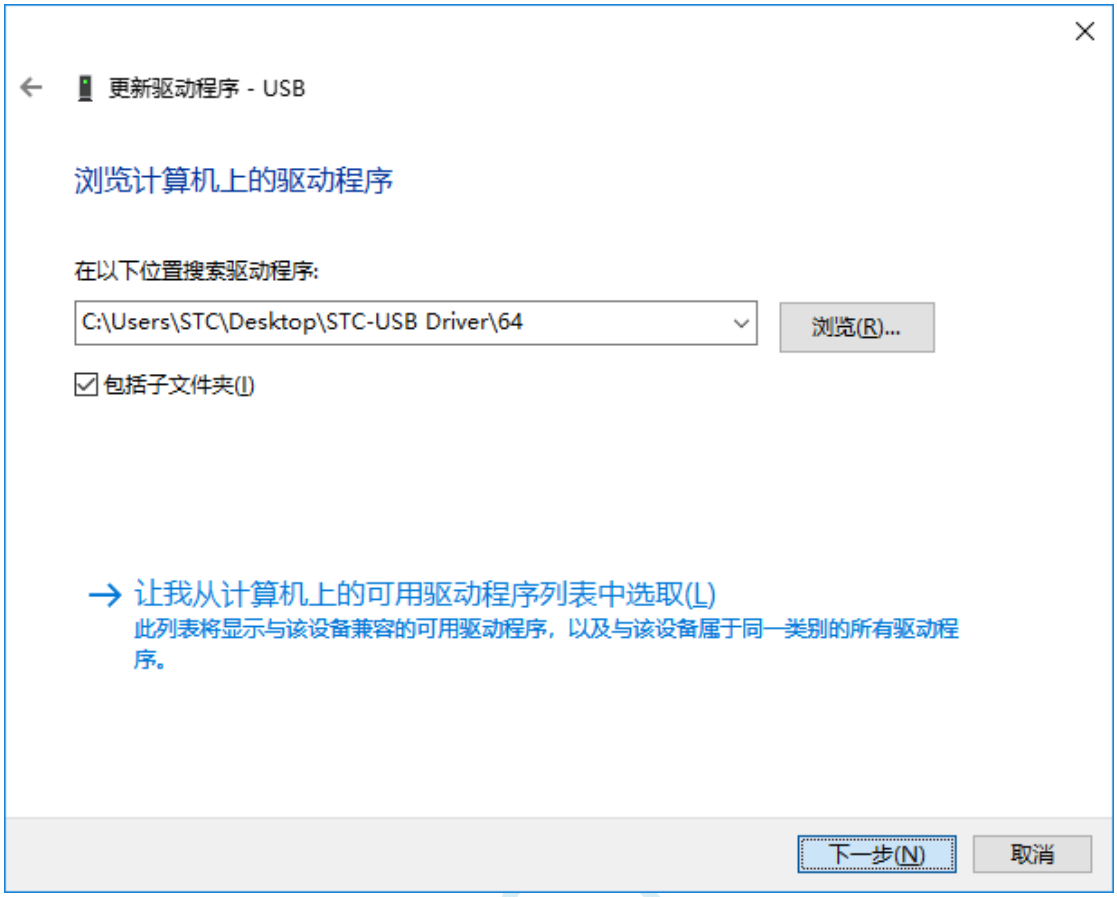
在如下界面中，点击“浏览”按钮



找到之前解压缩到硬盘中的“STC-USB Driver”目录，选择目录中的“64”目录，并确定



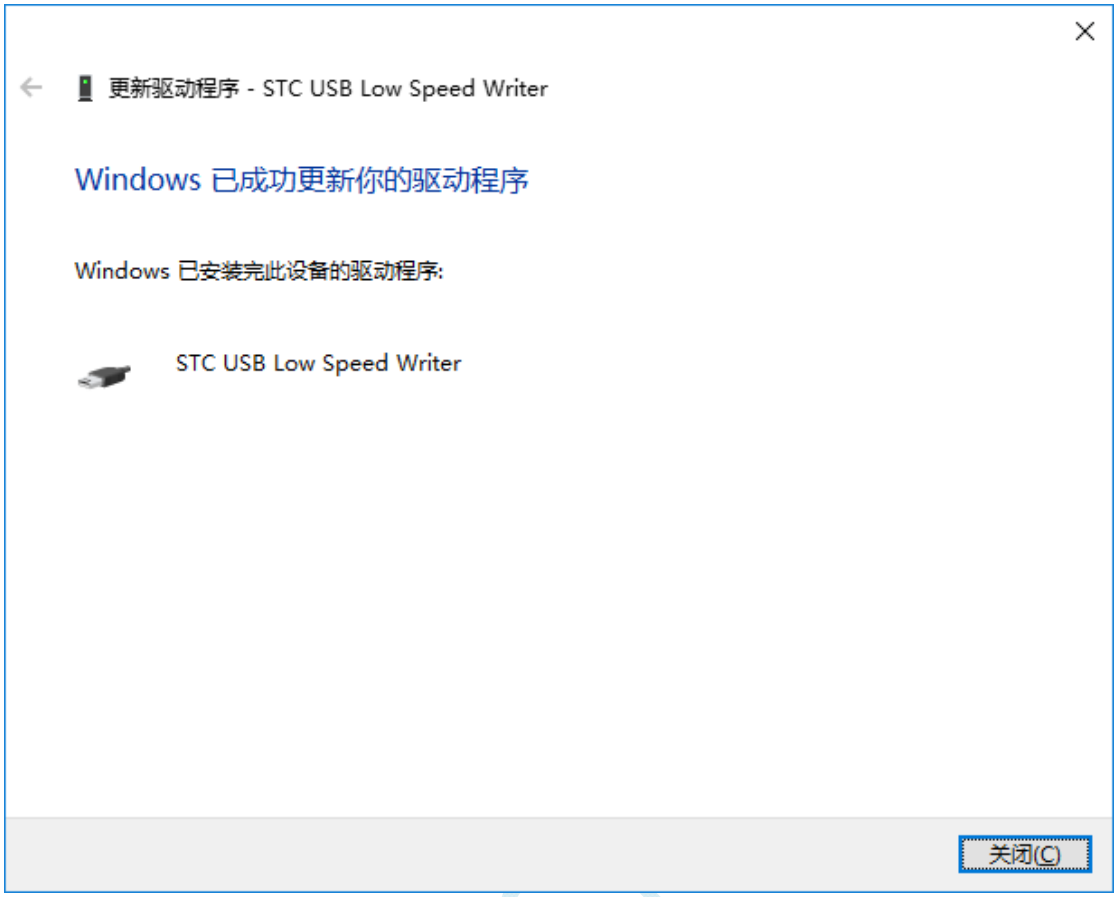
点击“下一步”开始安装驱动



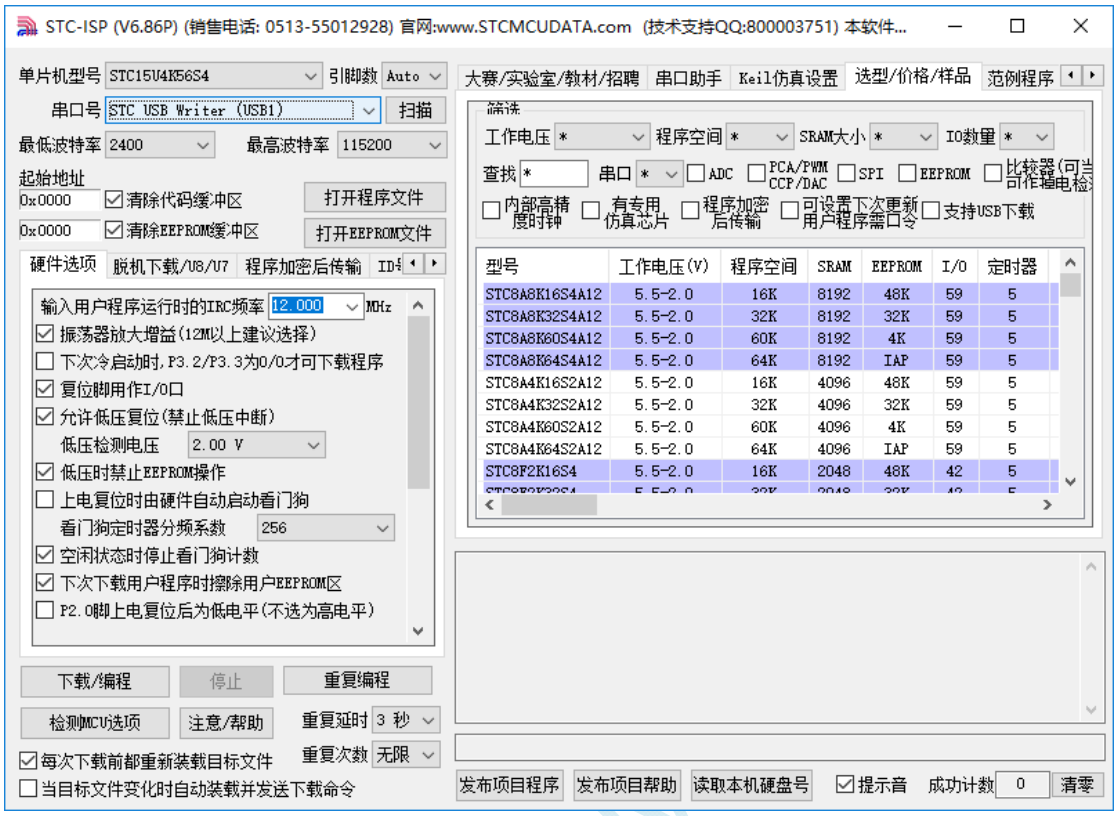
驱动安装的过程中，会弹出如下的警告画面，选择“始终安装此驱动程序软件”



出现下面的画面时，驱动程序就安装成功了

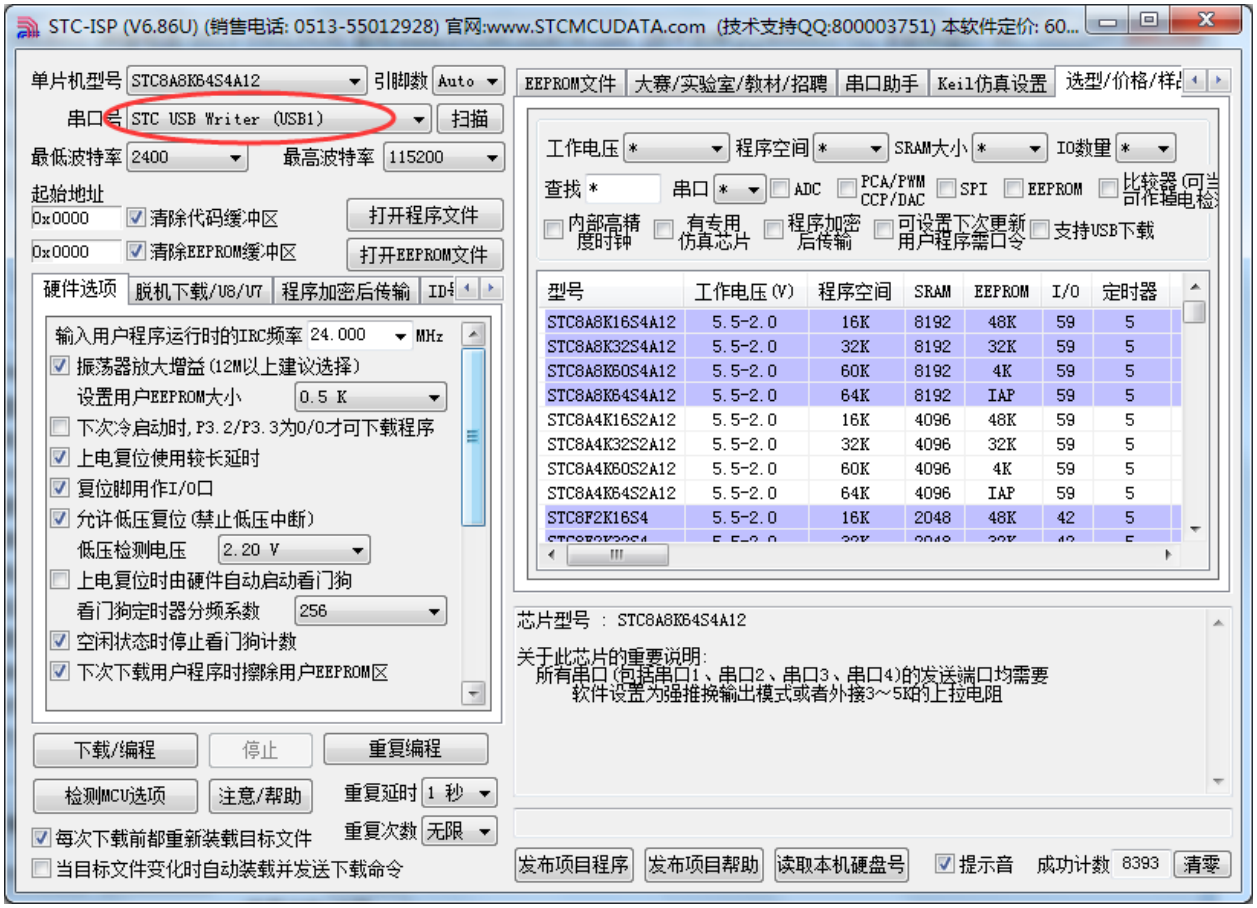


在回到 STC-ISP 的下载软件，此时“串口号”的下拉列表中已自动选择了“STC USB Writer (USB1)”，即可使用 USB 进行 ISP 下载了

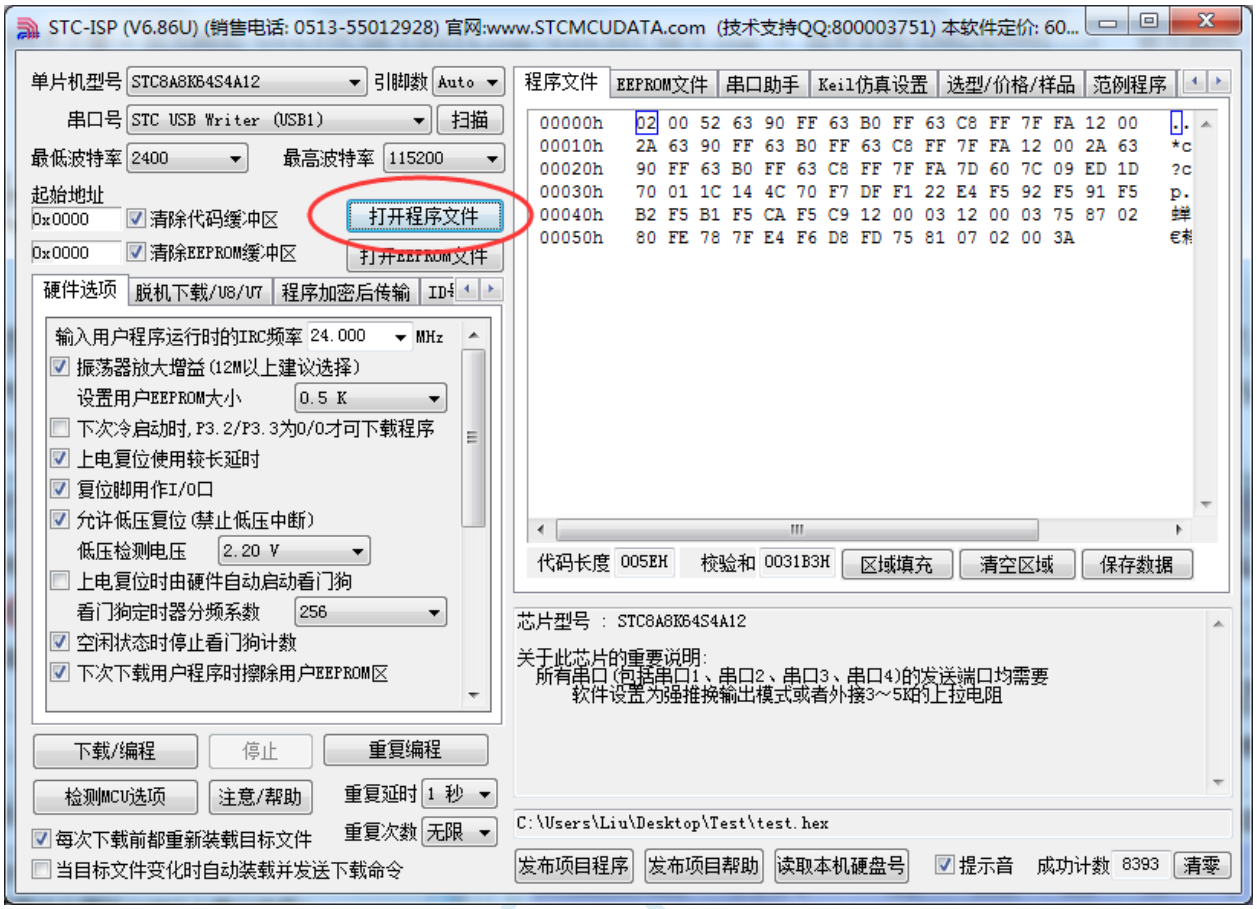


附录D USB下载步骤演示

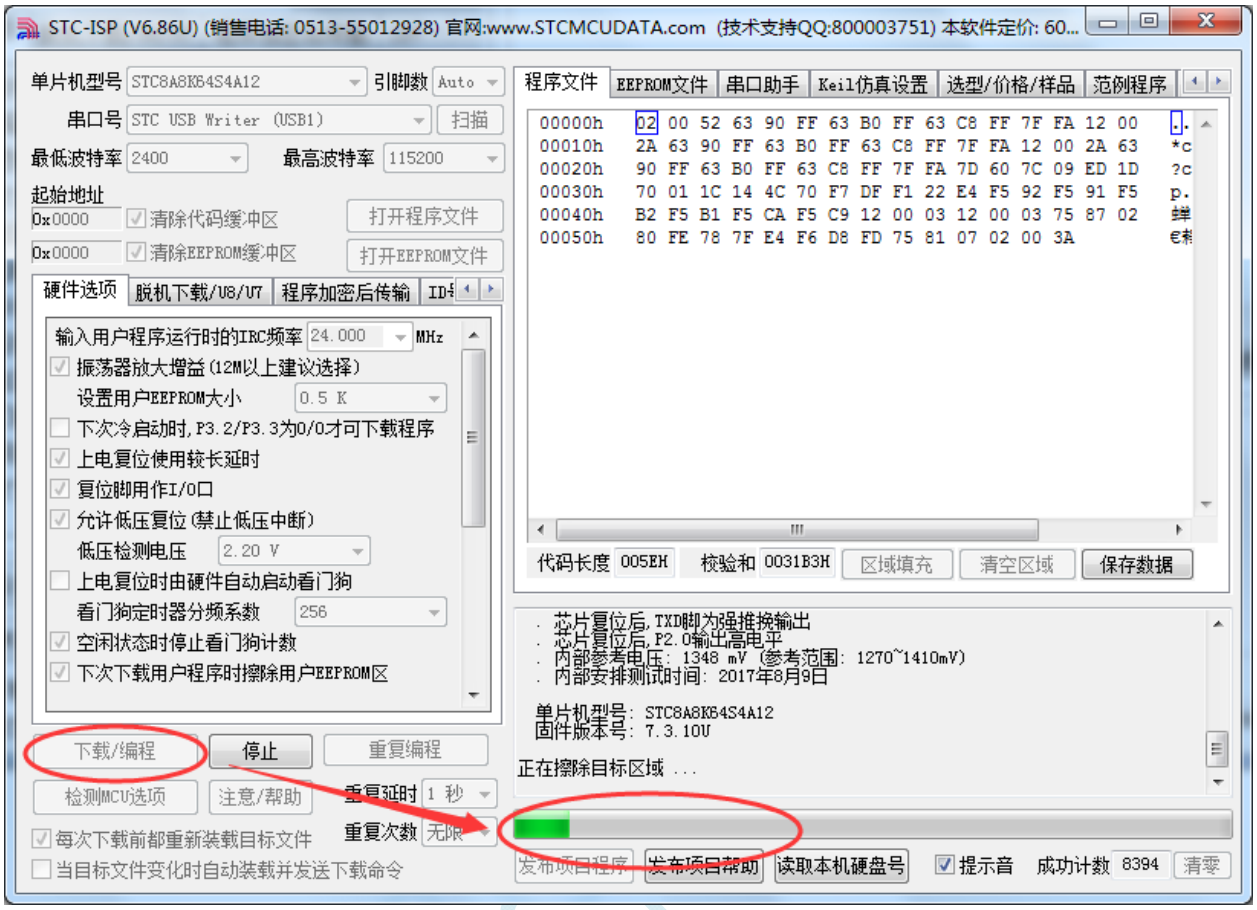
1、首先参考 P5.1.5 章的应用线路图连接好单片机，并将目标芯片的 P3.2 口连接到 Gnd，然后将系统连接到 PC 端的 USB 端口上。打开 ISP 下载软件，即可在下载软件的串口号中自动搜索到“STC USB Writer (USB1)”的 USB 设备



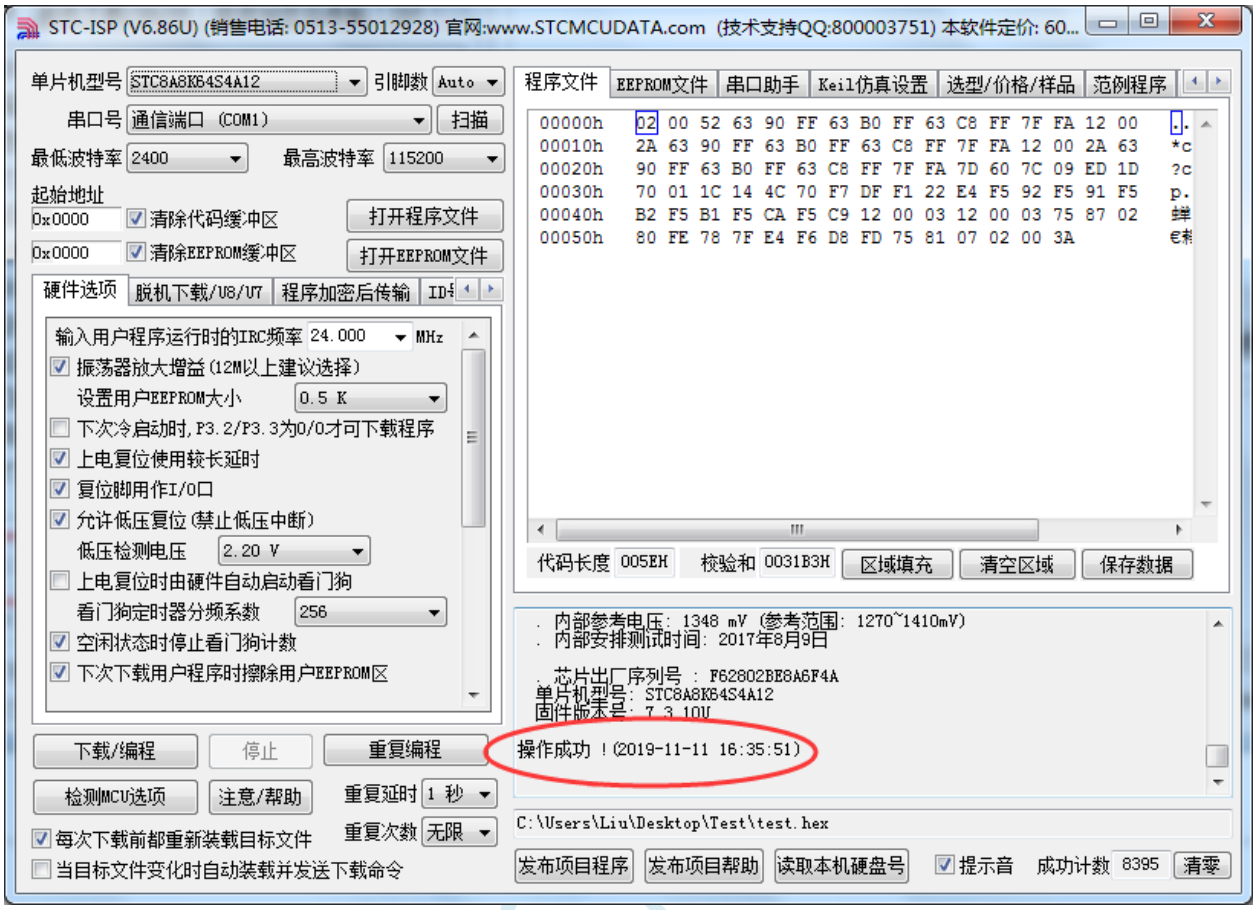
2、打开用户代码程序


STC-ISP (V6.86U) (销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (技术支持QQ:800003751) 本软件定价: 60...
单片机型号 STC8A8K64S4A12
引脚数 Auto
串口号 STC USB Writer (USB1)
扫描
最低波特率 2400
最高波特率 115200
起始地址
0x0000
清除代码缓冲区
打开程序文件
0x0000
清除EEPROM缓冲区
打开EEPROM文件
硬件选项
脱机下载/U8/U7
程序加密后传输
输入用户程序运行时的IRC频率 24.000 MHz
振荡器放大增益(12M以上建议选择)
设置用户EEPROM大小 0.5 K
下次冷启动时, P3.2/P3.3为0/0才可下载程序
上电复位使用较长延时
复位脚用作I/O口
允许低压复位(禁止低压中断)
低压检测电压 2.20 V
上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256
空闲状态时停止看门狗计数
下次下载用户程序时擦除用户EEPROM区
下载/编程
停止
重复编程
检测MCU选项
注意/帮助
重复延时 1 秒
每次下载前都重新装载目标文件
重复次数 无限
当目标文件变化时自动装载并发送下载命令
程序文件
EEPROM文件
串口助手
Keil仿真设置
选型/价格/样品
范例程序
00000h 02 00 52 63 90 FF 63 B0 FF 63 C8 FF 7F FA 12 00
00010h 2A 63 90 FF 63 B0 FF 63 C8 FF 7F FA 12 00 2A 63
00020h 90 FF 63 B0 FF 63 C8 FF 7F FA 7D 60 7C 09 ED 1D
00030h 70 01 1C 14 4C 70 F7 DF F1 22 E4 F5 92 F5 91 F5
00040h B2 F5 B1 F5 CA F5 C9 12 00 03 12 00 03 75 87 02
00050h 80 FE 78 7F E4 F6 D8 FD 75 81 07 02 00 3A
代码长度 005EH
校验和 0031B3H
区域填充
清空区域
保存数据
芯片型号 : STC8A8K64S4A12
关于此芯片的重要说明:
所有串口(包括串口1、串口2、串口3、串口4)的发送端口均需要
软件设置为强推挽输出模式或者外接3~5K的上拉电阻
C:\Users\Liu\Desktop\Test\test.hex
发布项目程序
发布项目帮助
读取本机硬盘号
提示音
成功计数 8393
清零

3、点击“下载/编程”按钮开始下载用户代码

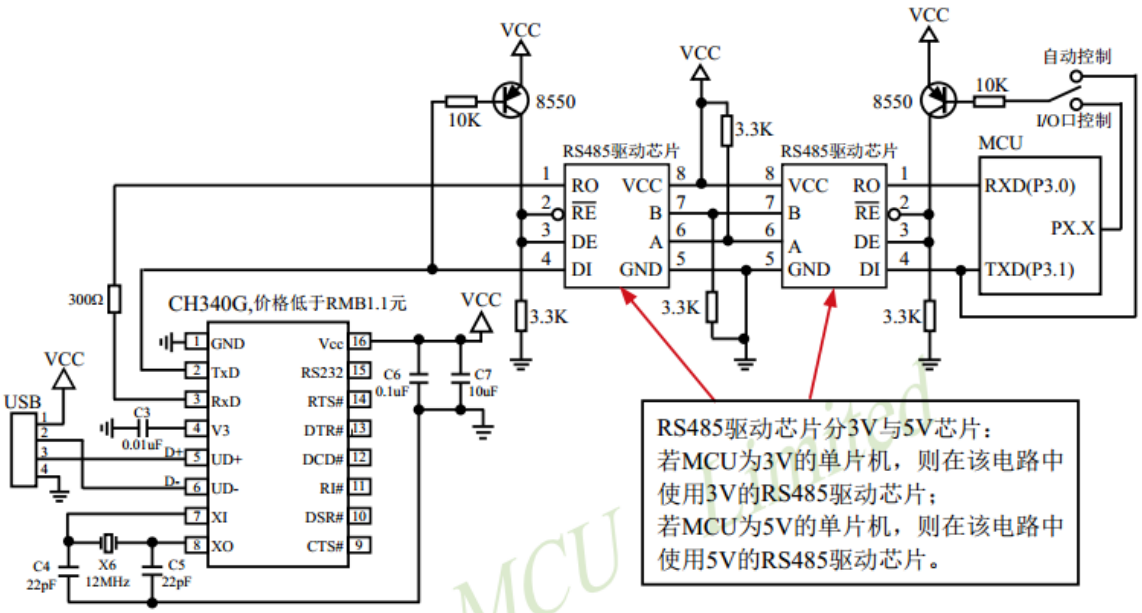


4、直到提示“操作成功”，表示程序代码下载完成。

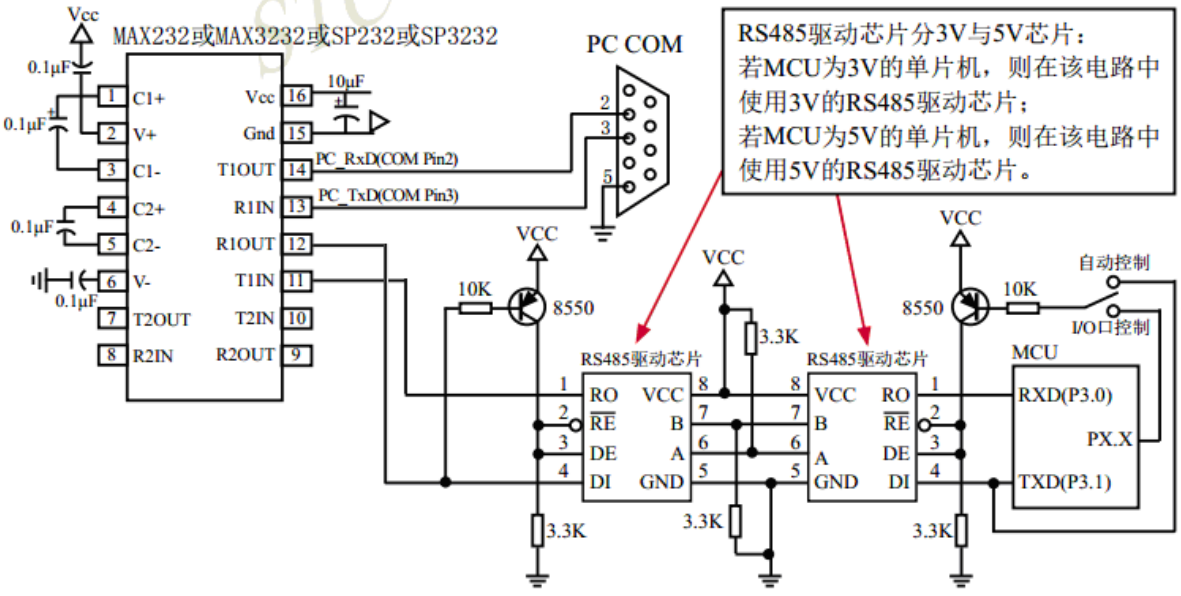


附录E RS485 自动控制或I/O口控制线路图

1、利用 USB 转串口连接电脑的 RS485 控制下载线路图(自动控制或 I/O 口控制)



2、利用 RS232 转串口连接电脑的 RS485 控制下载线路图(自动控制或 I/O 口控制)



注意：如果要设置单片机某个I/O口控制RS485发送或接收命令有效，则必须将单片机焊入电路板之前先用U8下载工具结合电脑ISP软件对该单片机进行“RS485控制”设置并烧录一下（如上节所述），否则将单片机实现不了RS485控制功能。

建议用户将本节所述“RS485控制下载线路图(自动控制或I/O口控制)”设计到您的用户板上

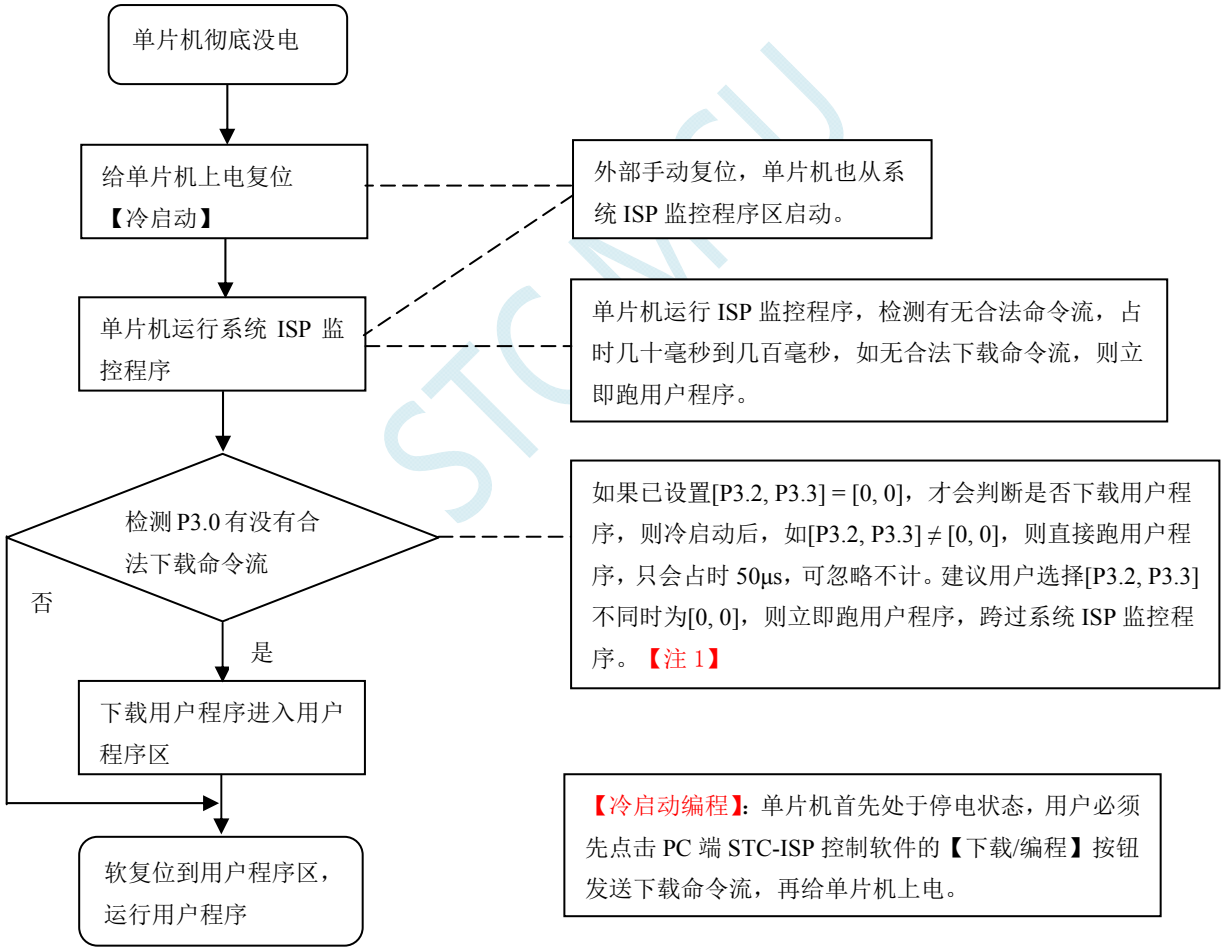
附录F　STC工具使用说明书

F.1　概述

U8W/U8W-Mini 是一款集在线联机下载和脱机下载于一体的编程工具系列。STC 通用 USB 转串口工具则是支持在线下载与在线仿真的编程工具。

工具类型	在线下载	脱机下载	烧录座下载	在线仿真	价格(人民币)
U8W	支持	支持	支持	需设置直通模式	100 元
U8W-Mini	支持	支持	不支持	需设置直通模式	50 元
通用 USB 转串口	支持	不支持	不支持	支持	30 元

F.2　系统可编程（ISP）流程说明

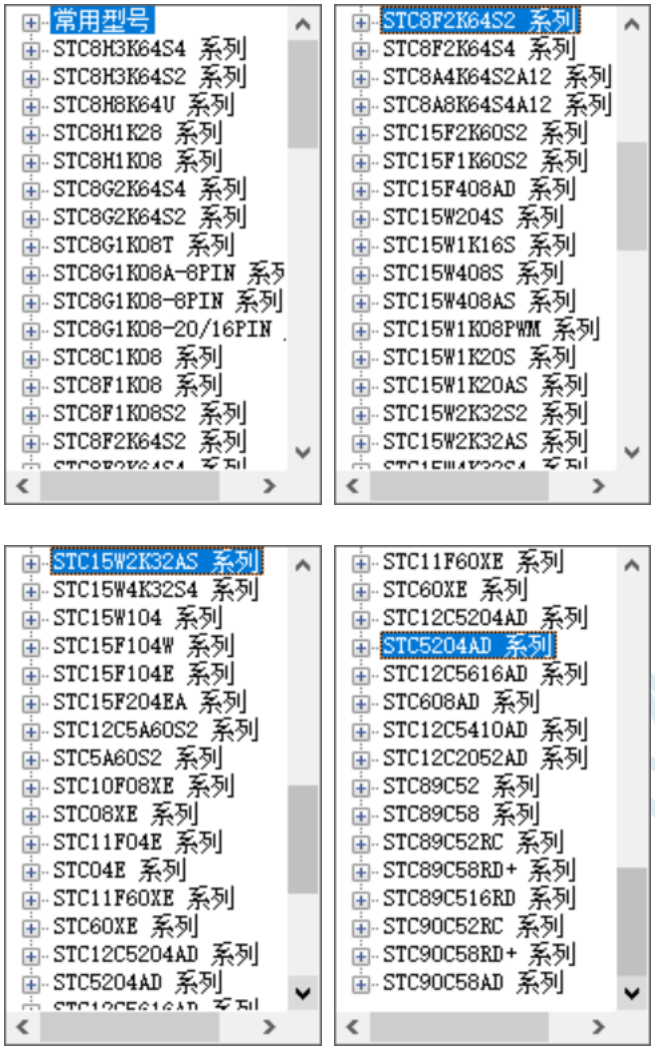


注意：因 [P3.0, P3.1] 作下载/仿真用(下载/仿真接口仅可用 [P3.0, P3.1])，故建议用户将串口 1 放在 P3.6/P3.7 或 P1.6/P1.7 ，若用户不想切换，坚持使用 P3.0/P3.1 工作或作为串口 1 进行通信，则务必在下载程序时，在软件上勾选“下次冷启动时，P3.2/P3.3 为 0/0 时才可以下载程序”。【注 1】

【注 1】：STC15，STC8 系列及以后新出的芯片的烧录保护引脚为 P3.2/P3.3，之前早期芯片的烧录保护引脚为 P1.0/P1.1。

F.3 USB型联机/脱机下载工具U8W/U8W-Mini

U8W/U8W-Min 的应用范围可支持 STC 目前的全部系列的 MCU, Flash 程序空间和 EEPROM 数据空间不受限制。支持包括如下和即将推出的 STC 全系列芯片：



脱机下载工具可以在脱离电脑的情况下进行下载工作，可用于批量生产和远程升级。脱机下载板可支持自动增量、下载次数限制以及用户程序加密后传输等多种功能。

下图为 U8W 工具的正反面图以及 U8W-Mini 的正反面图：



U8W正面图



U8W反面图


U8W-Mini正反面图

U8W-Mini工具的体积仅有一个U盘大小，其功能与U8W相同，但无锁紧座，价格仅为RMB 50元，欢迎来电订购！

另外还有如下的一些线材与工具相搭配使用，如：

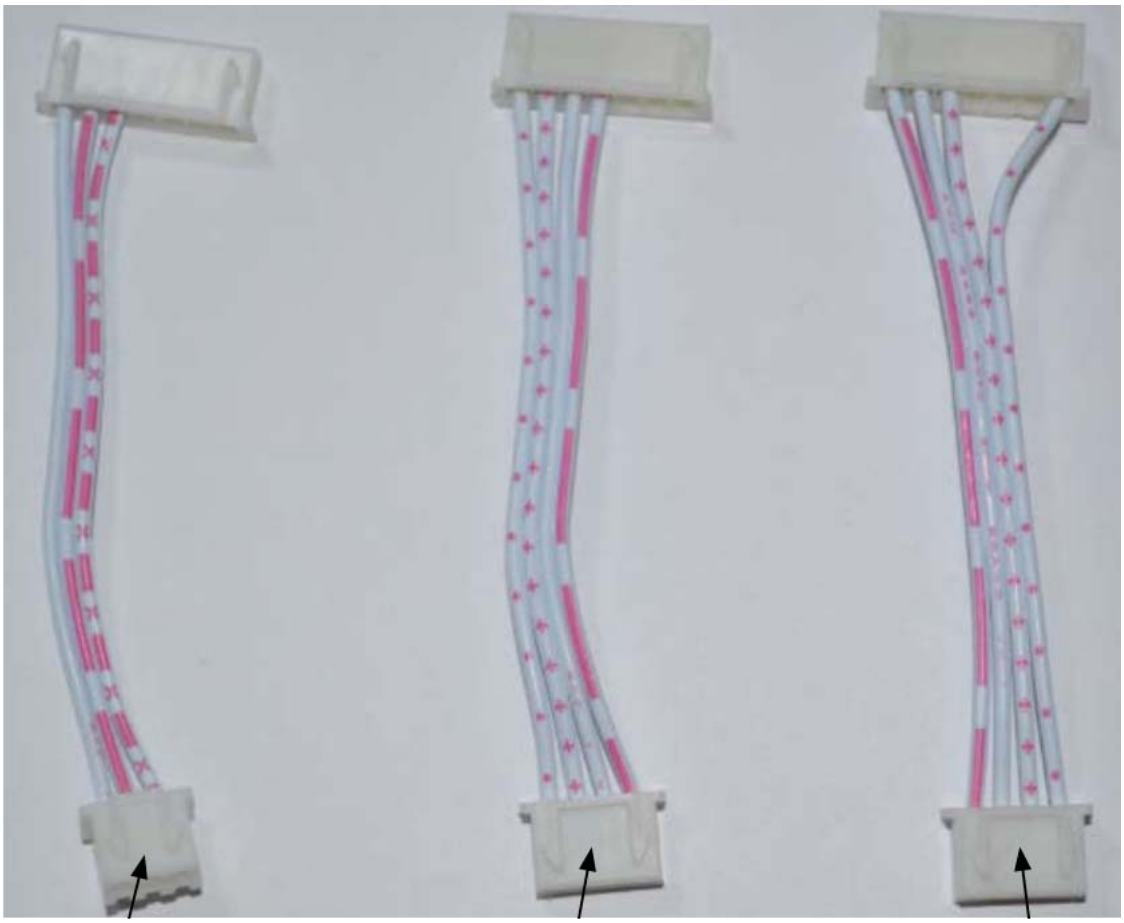
（1）两头公的 USB 连接线(如下图左所示）及 USB-Micro 连接线(如下图右所示)：





注意：此 USB 线为我公司特别定制的 USB 加强线，可确保直接用 USB 供电时能够下载成功。而市面上一些比较劣质的两头公的 USB 线，内阻太大而导致压降很大（如 USB 空载时的电压为 5.0V 左右，当使用劣质的 USB 线连接 U8W/U8W-Mini/U8/U8-Mini，到我们的下载板上的电压可能降到 4.2V 或者更低，从而导致芯片处于复位状态而无法成功下载）。

（2）U8W/U8W-Mini 与用户系统连接的下载连接线(即 U8W/U8W-Mini 与用户板上的目标单片机的连接线)，如下图所示：



U8W/U8W-Mini 与用户系统各自独立供电的连接线

U8W/U8W-Mini 给用户系统供电的连接线

用户系统给 U8W/U8W-Mini 供电的连接线

F.3.1 安装U8W/U8W-Mini驱动程序

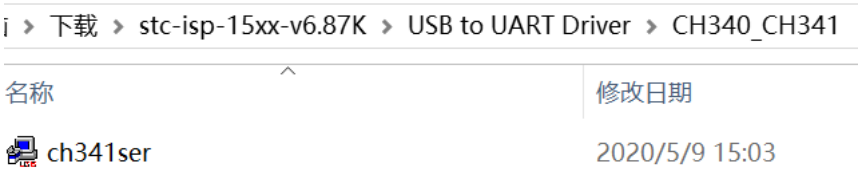
U8W/U8W-Mini 下载板上使用了一颗 CH340 的 USB 转串口通用芯片。这样可以省去部分没有串口的电脑必须额外买一个 USB 转串口工具才可下载的麻烦。但 CH340 和其它 USB 转串口工具一样，在使用之前必须先安装驱动程序。

通过下载 STC-ISP 软件包获取驱动程序

以下是 STC 官网（www.STCMCUDATA.com）提供的 STC-ISP 软件包下载位置：

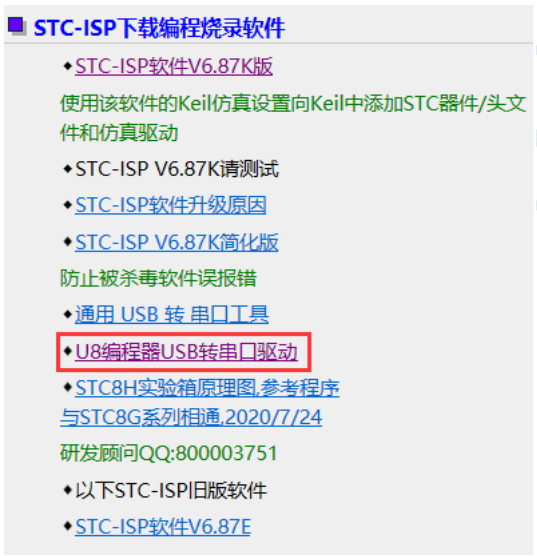


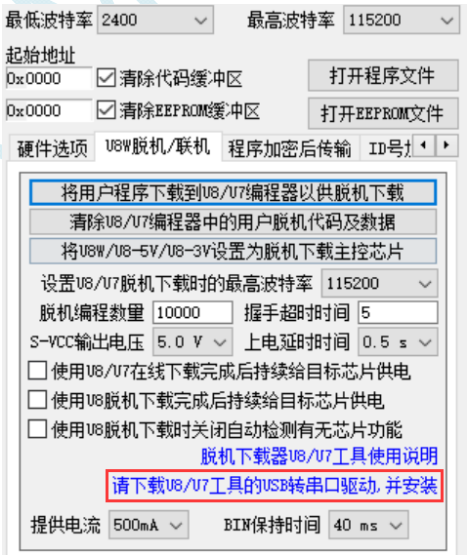
下载后进行解压，CH340 的驱动安装包路径 stc-isp-15xx-v6.87K\USB to UART Driver\CH340_CH341：



通过 STC 的官方网站或在最新的 STC-ISP 下载软件中手动下载驱动程序

在 STC 的官方网站上或在最新的 STC-ISP 下载软件中手动下载驱动程序，驱动的下载链接为： U8 编程器 USB 转串口驱动 （ http://www.stcmcu.com/STCISP/CH341SER.exe）。网站上及 STC-ISP 下载软件上的驱动地址如下图所示：



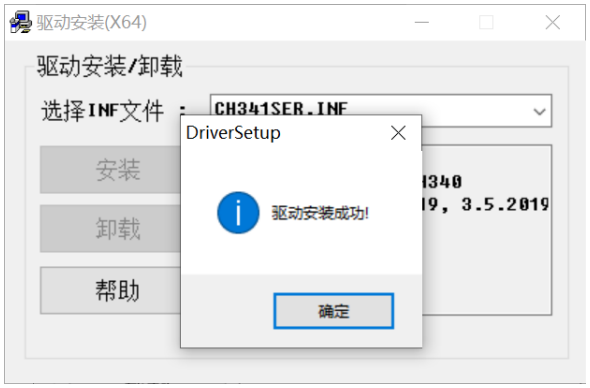


安装 U8W/U8W-Mini 的驱动程序

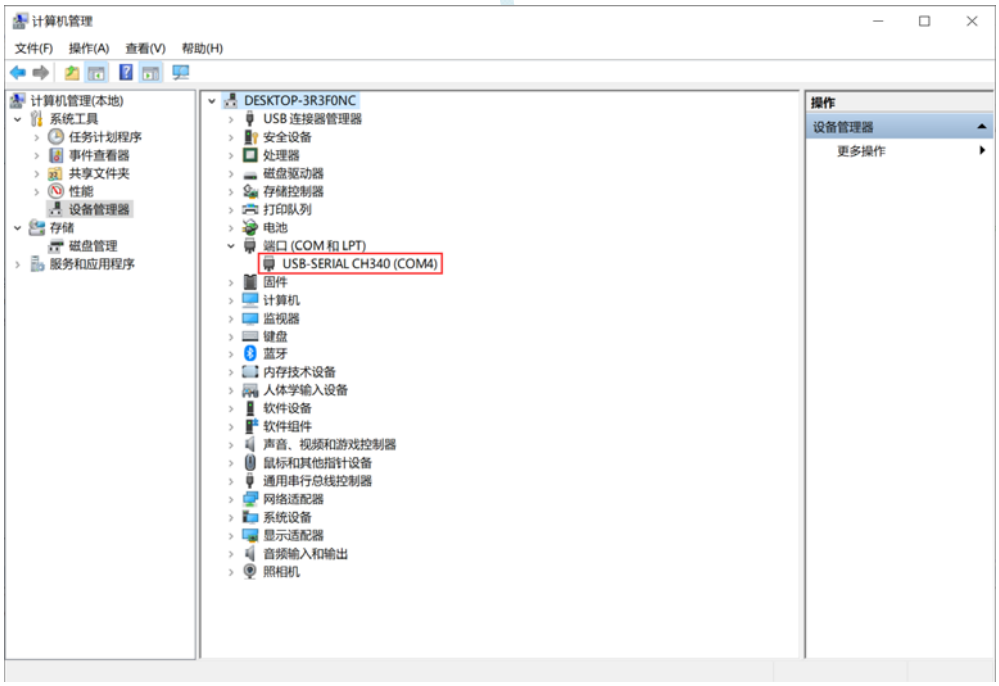
驱动程序下载到本机后，直接双击可执行程序并运行，出现下图所示的界面，点击“安装”按钮开始自动安装驱动：



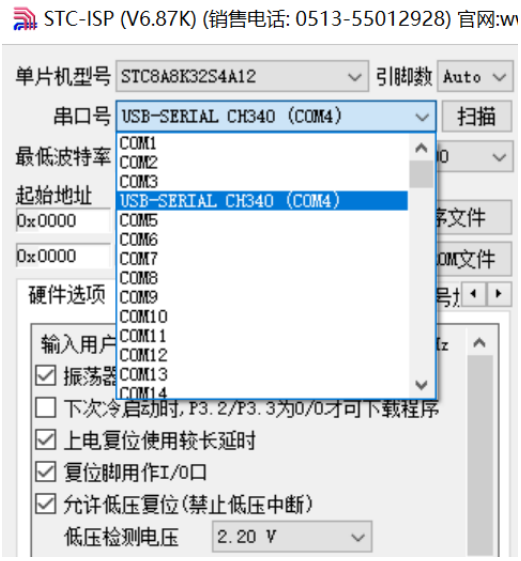
然后弹出驱动安装成功对话框，点击“确定”按钮完成安装：



然后使用 STC 提供的 USB 连接线将 U8W/U8W-Mini 下载板连接到电脑，打开电脑的设备管理器，在端口设备类下面，如果有类似“USB-SERIAL CH340 (COMx)”的设备，就表示 U8W/U8W-Mini 可以正常使用了。如下图所示（不同的电脑，串口号可能会不同）：



注意：在后面使用 STC-ISP 下载软件时，选择的串口号必须选择与此相对应的串口号，如下图所示：

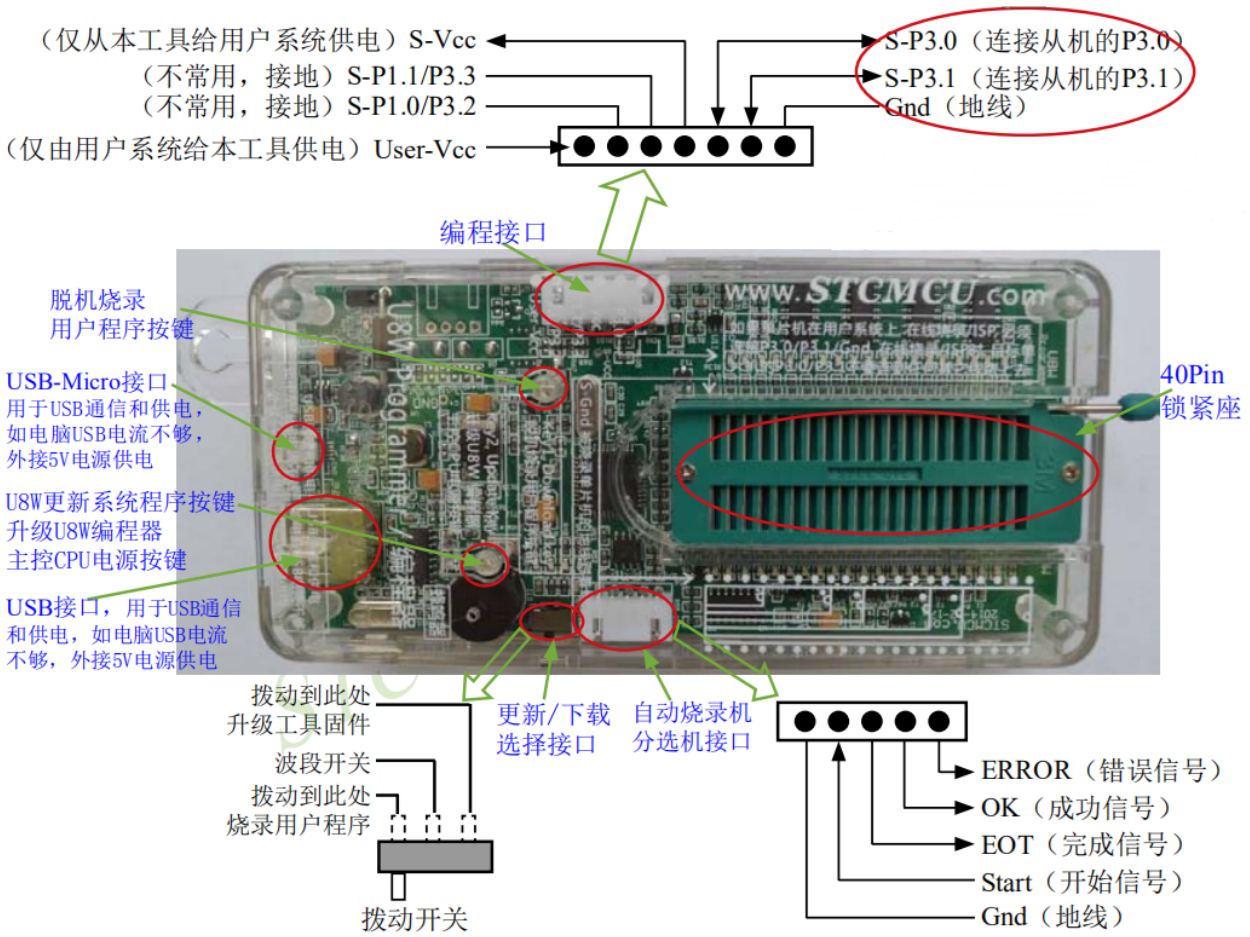


F.3.2 U8W的功能介绍

下面详细介绍 U8W 工具的各主要接口及功能：

如果单片机在用户系统上，在线烧录/ISP 时必须连接 P3.0/P3.1/Gnd，在线烧录/ISP 时，目标单片机的 P3.0/P3.1 不要连到任何其他线路上去

STC MCU



编程接口：根据不同的供电方式，使用不同的下载连接线连接 U8W 下载板和用户系统。

U8W 更新系统程序按键：用于更新 U8W 工具，当有新版本的 U8W 固件时，需要按下此按键对 U8W 的主控芯片进行更新（注意：必须先将更新/下载选择接口上的拨动开关拨动到升级工具固件）。

脱机下载用户程序按钮：开始脱机下载按钮。首先 PC 将脱机代码下载到 U8W 板上，然后使用下载连接线将用户系统连接到 U8W，再按下此按钮即可开始脱机下载（每次上电时也会立即开始下载用户代码）。

更新/下载选择接口：当需要对 U8W 的底层固件进行升级时，需将此拨动开关拨到升级工具固件处，当需通过 U8W 对目标芯片进行烧录程序，则需将拨动开关拨到烧录用户程序处。

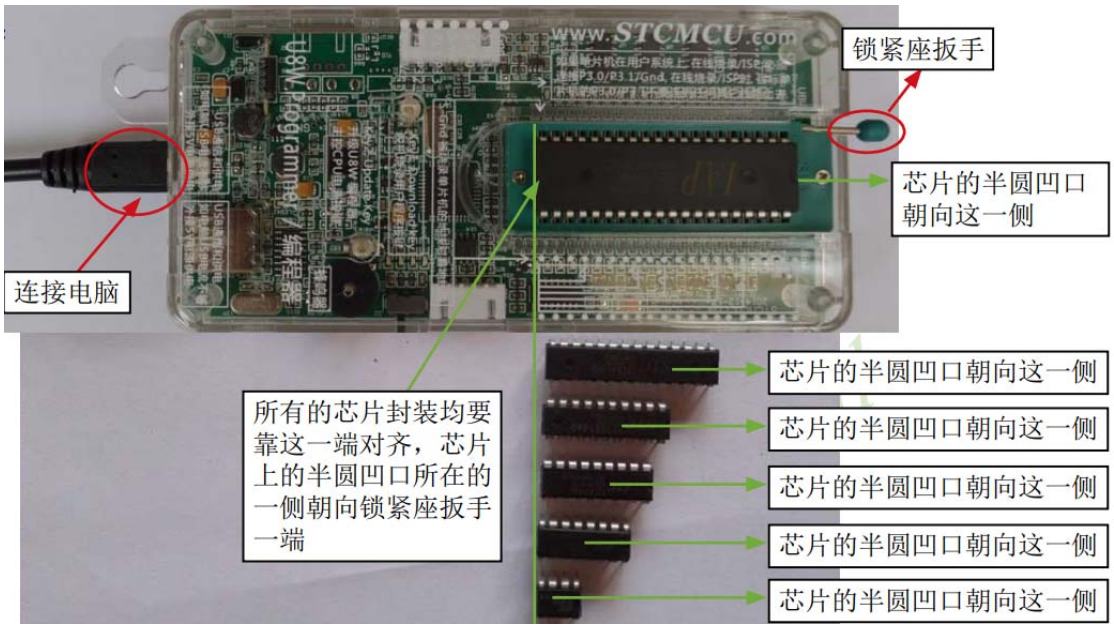
（拨动开关连接方式请参考上图）

自动烧录机/分选机接口：是用于控制自动烧录机/分选机进行自动生产的控制接口。

F.3.3 U8W的在线联机下载使用说明

目标芯片安装于 U8W 锁紧座上并由 U8W 连接电脑进行在线联机下载

首先使用 STC 提供的 USB 连接线将 U8W 连接电脑，再将目标单片机按如下图所示的方向安装在 U8W 上：



然后使用 STC-ISP 下载软件下载程序，步骤如下：



1 选择单片机型号；

2 选择引脚数，芯片直接安装于 U8W 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；

3 选择 U8W 所对应的串口号；

4 打开目标文件（HEX 格式或者 BIN 格式）；

5 设置硬件选项；

6 点击“下载/编程”按钮开始烧录；

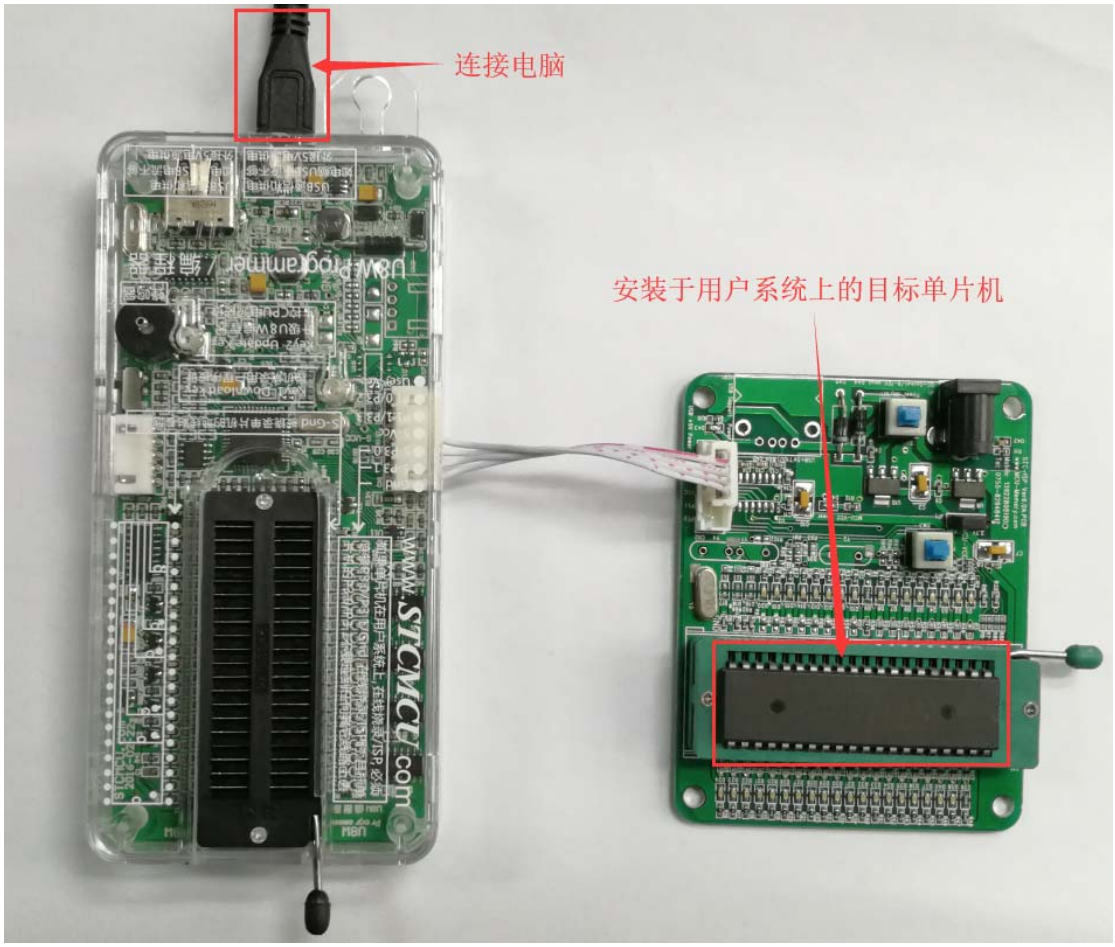
7 显示烧录过程的步骤信息，烧录完成提示“操作成功！”。

当信息框中有输出下载板的版本号信息以及外挂 Flash 的相应信息时，表示已正确检测到 U8W 下载工具。

下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。
建议用户用最新版本的 STC-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCMCUDATA.com 中 STC-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCMCUDATA.com 中下载最新版本的软件使用）。

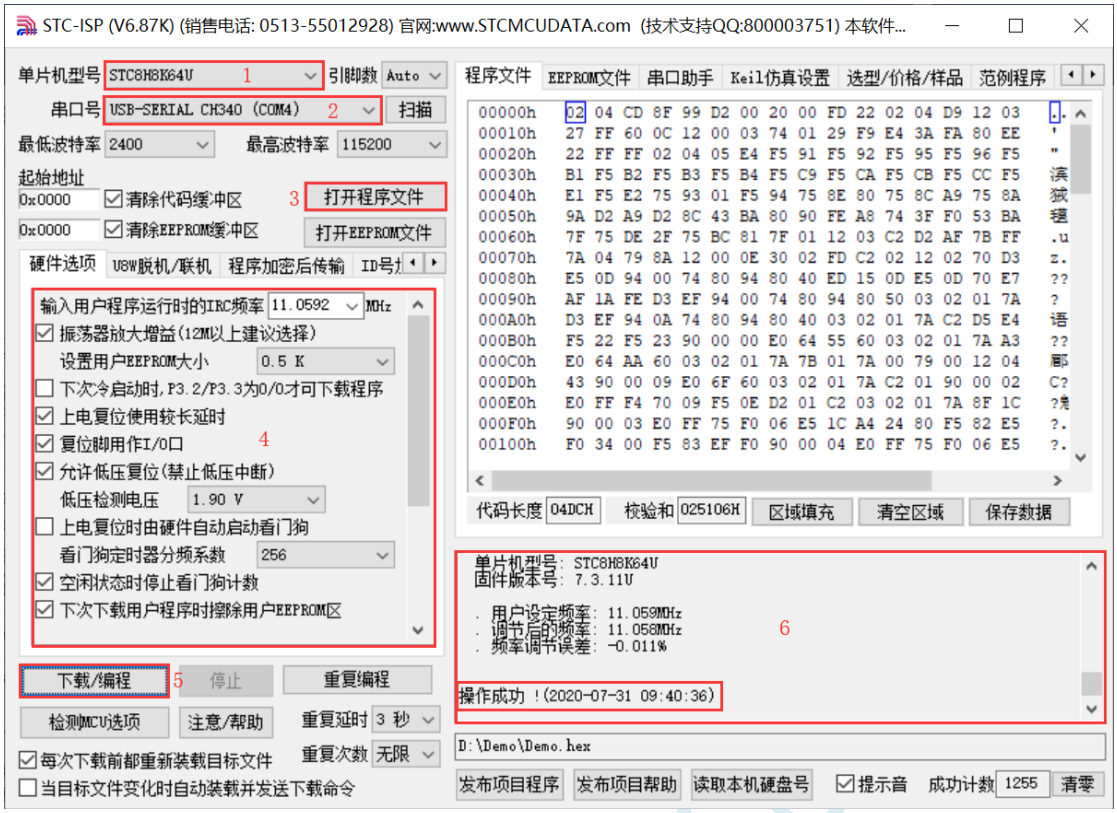
目标芯片通过用户系统引线连接 U8W 并由 U8W 连接电脑进行在线联机下载

首先使用 STC 提供的 USB 连接线将 U8W 连接电脑，再将 U8W 通过下载线与用户系统的目标单片机相连接，连接方式如下图所示：



然后使用 STC-ISP 下载软件下载程序，步骤如下：

1. 选择单片机型号；
2. 选择 U8W 所对应的串口号；
3. 打开目标文件（HEX 格式或者 BIN 格式）；
4. 设置硬件选项；
5. 点击“下载/编程”按钮开始烧录；
6. 显示烧录过程的步骤信息，烧录完成提示“操作成功！”。



当信息框中有输出下载板的版本号信息以及外挂 Flash 的相应信息时，表示已正确检测到 U8W 下载工具。下载的过程中， U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

建议用户用最新版本的 STC-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCMCUDATA.com 中 STC-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCMCUDATA.com 中下载最新版本的软件使用）。

F.3.4 U8W的脱机下载使用说明

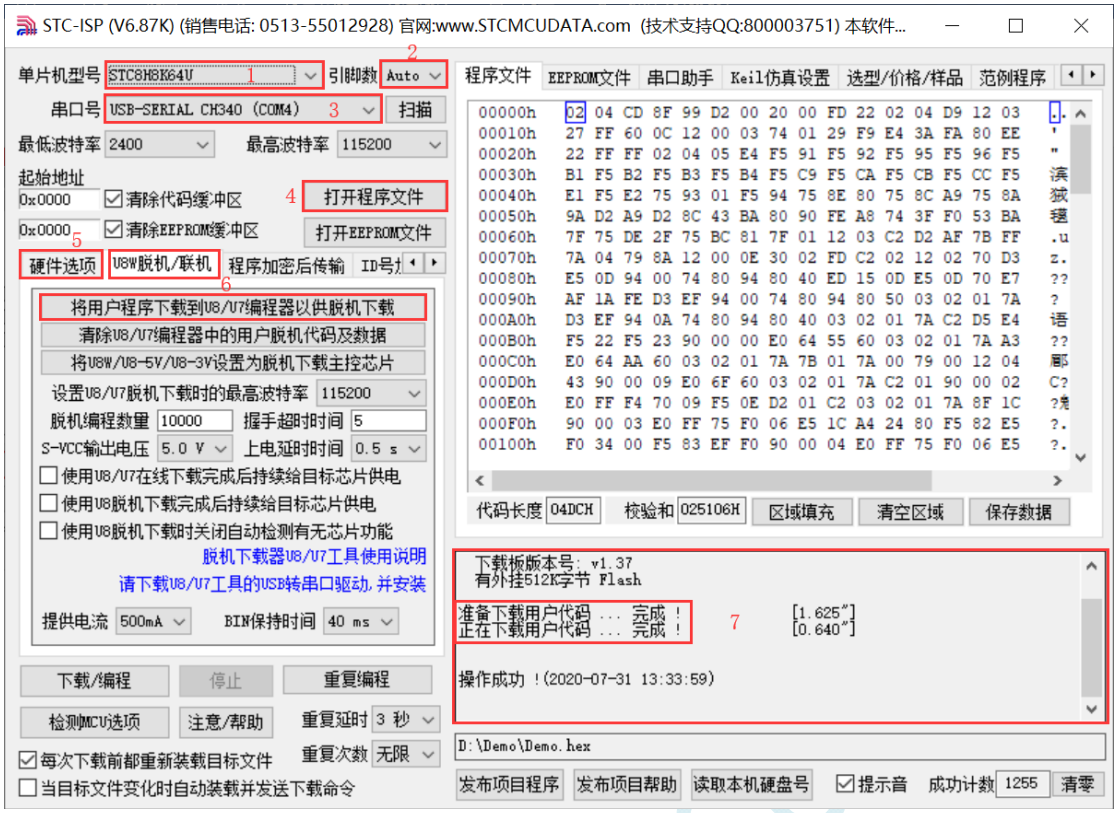
目标芯片安装于 U8W 座锁紧上并通过 USB 连接电脑给 U8W 供电进行脱机下载

使用 USB 给 U8W 供电从而进行脱机下载的步骤如下：

（1）使用 STC 提供的 USB 连接线将 U8W 下载板连接到电脑，如下图：



（2）在 STC-ISP 下载软件中按如下图所示的步骤进行设置：

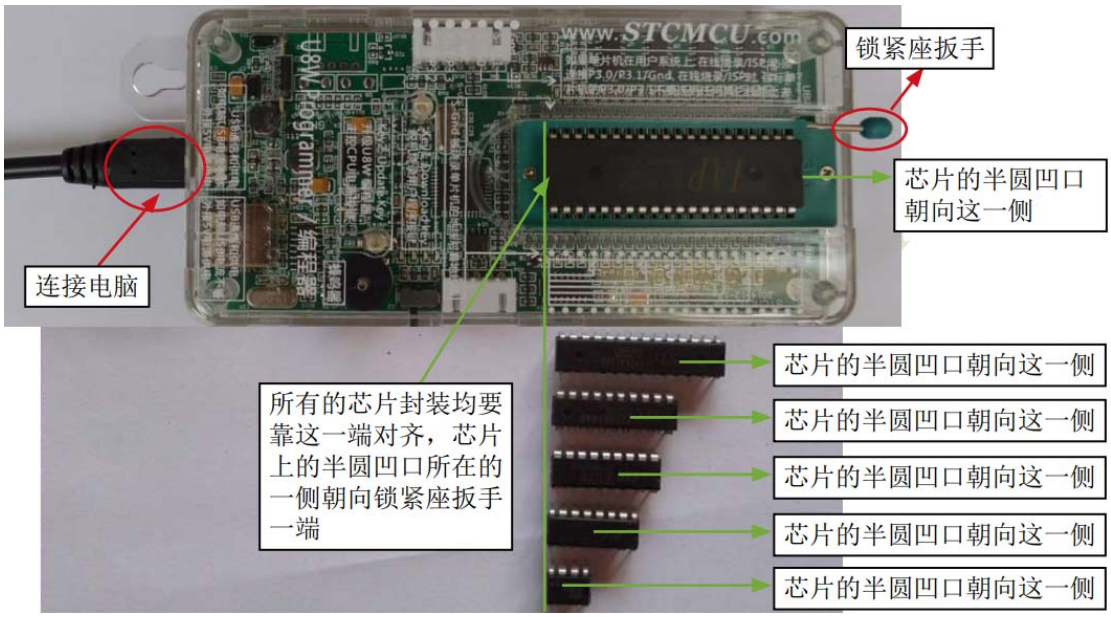


1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

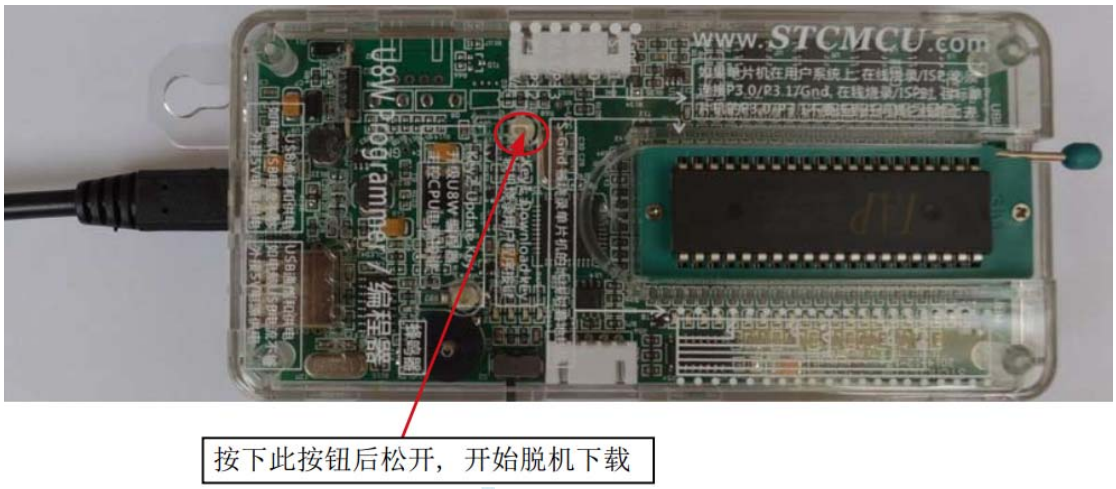
按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8W 下载工具中。

建议用户用最新版本的 STC-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCMCUDATA.com 中 STC-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCMCUDATA.com 中下载最新版本的软件使用）。

（3）再将目标单片机如下图所示的方向放在 U8W 下载工具，如下图所示：



（4）然后按下如下图所示的按钮后松开，即可开始脱机下载：



下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

脱机下载即插即用烧录功能介绍：

1. 以上步骤完成(1)、(2)步之后 U8W 连接电脑上电时默认处于即插即用烧录状态；
2. 按照第(3)步指示将芯片放入烧录座，在锁紧座扳手的同时，U8W 会自动开始烧录；
3. 通过指示灯显示烧录过程跟烧录结果；
4. 烧录完成后松开座扳手，取出芯片；
5. 重复 2，3，4 步骤可进行连续烧录，省掉按按钮触发烧录的动作。

目标芯片由用户系统引线连接 U8W 并通过 USB 连接电脑给 U8W 供电进行脱机下载

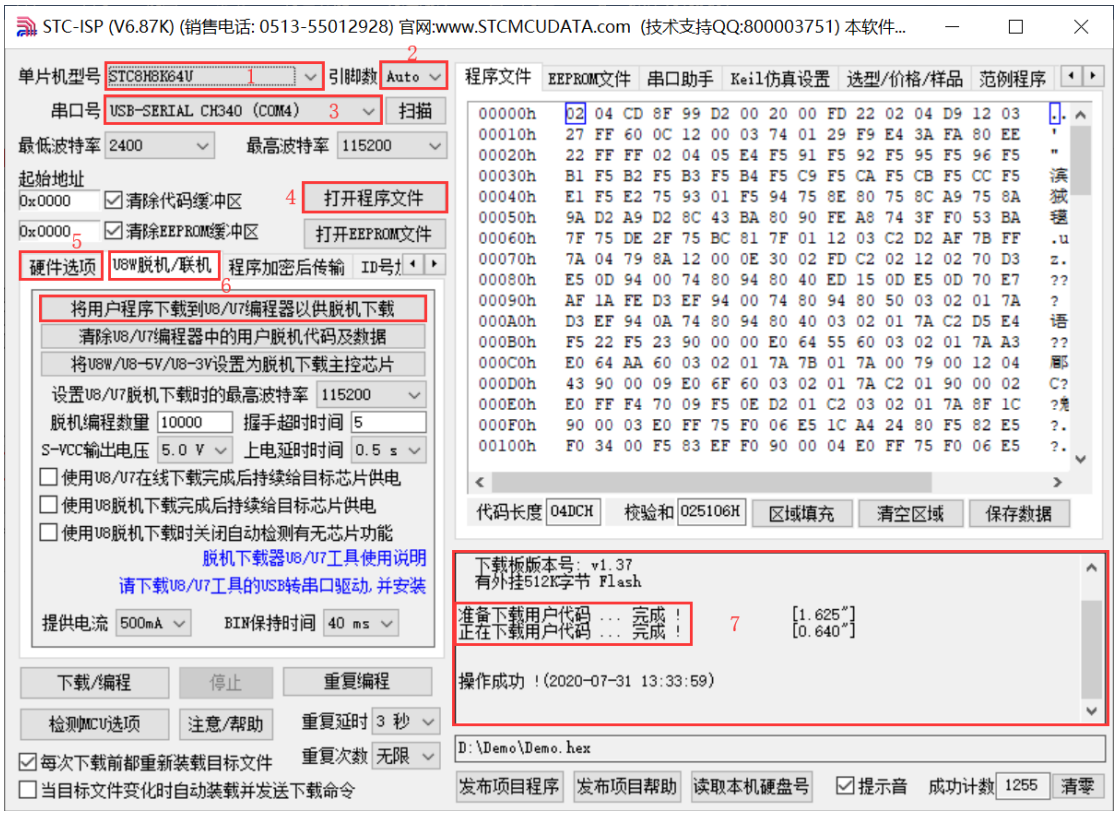
使用 USB 给 U8W 供电从而进行脱机下载的步骤如下：

（1）使用 STC 提供的 USB 连接线将 U8W 下载板连接到电脑，如下图：



（2）在 STC-ISP 下载软件中按如下图所示的步骤进行设置：

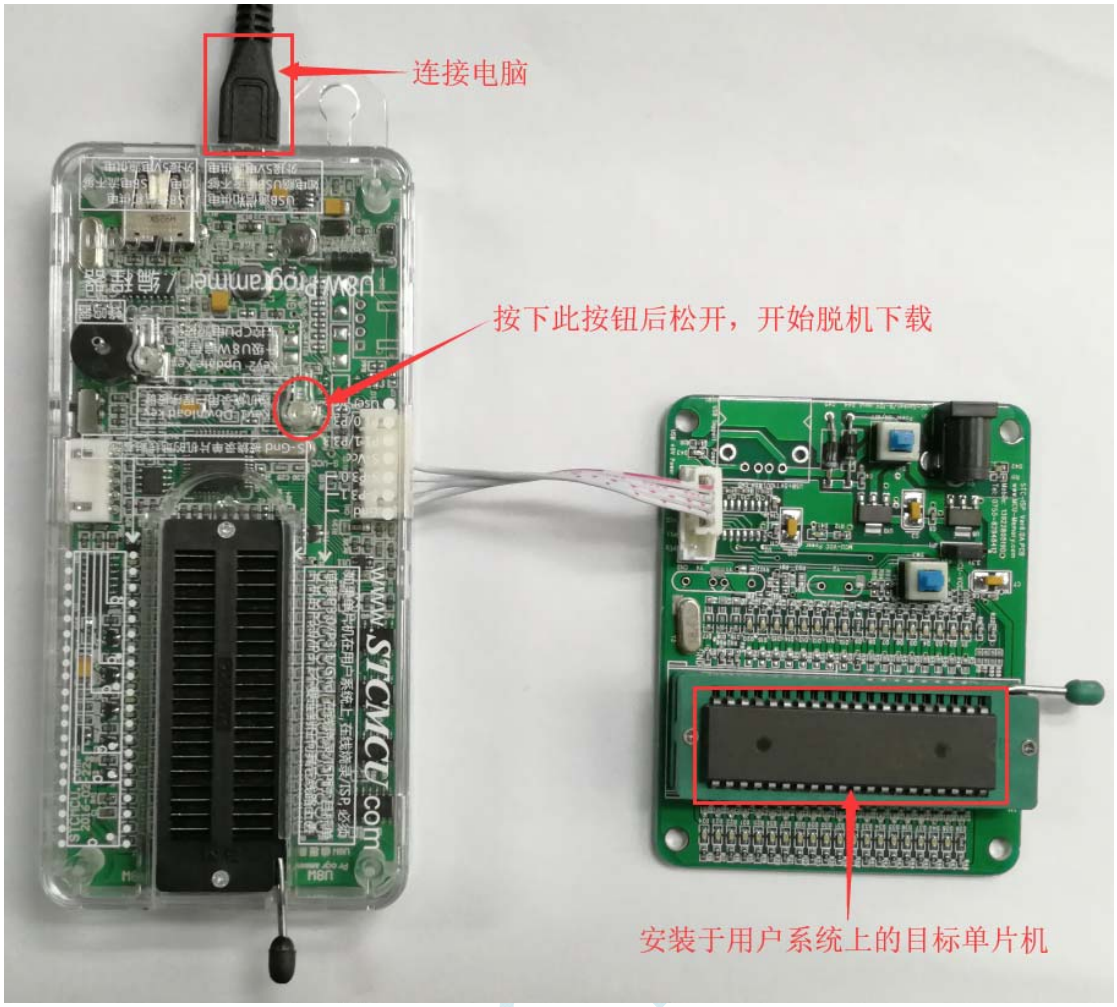
建议用户用最新版本的 STC-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCMCUDATA.com 中 STC-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCMCUDATA.com 中下载最新版本的软件使用）。



1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8W 下载工具中。

（3）然后使用连接线连接电脑、将 U8W 下载工具以及用户系统（目标单片机）如下图所示的方式连接起来，并按下图所示的按钮后松开，即可开始脱机下载：



下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

目标芯片由用户系统引线连接 U8W 并通过用户系统给 U8W 供电进行脱机下载

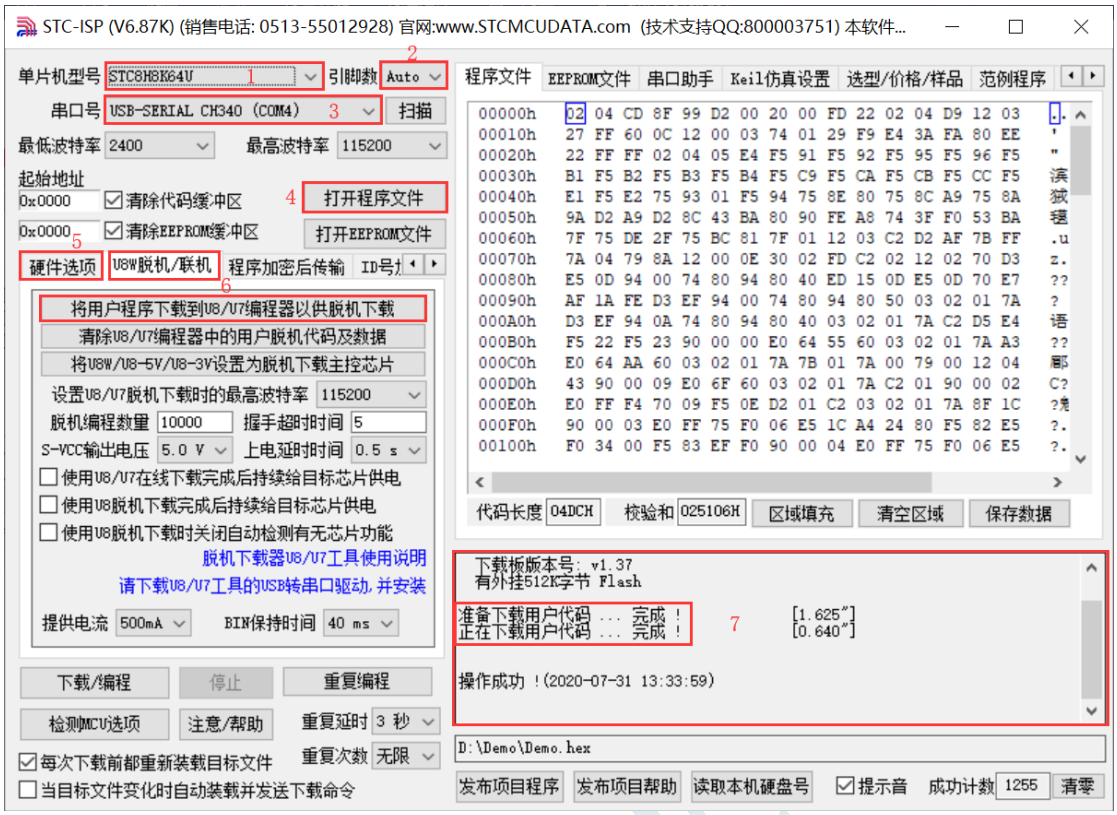
（1）首先使用 STC 提供的 USB 连接线将 U8W 下载板连接到电脑，如下图：



（2）在 STC-ISP 下载软件中按如下图所示的步骤进行设置：

建议用户用最新版本的 STC-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCMCUDATA.com 中 STC-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCMCUDATA.com 中下载最新版本的

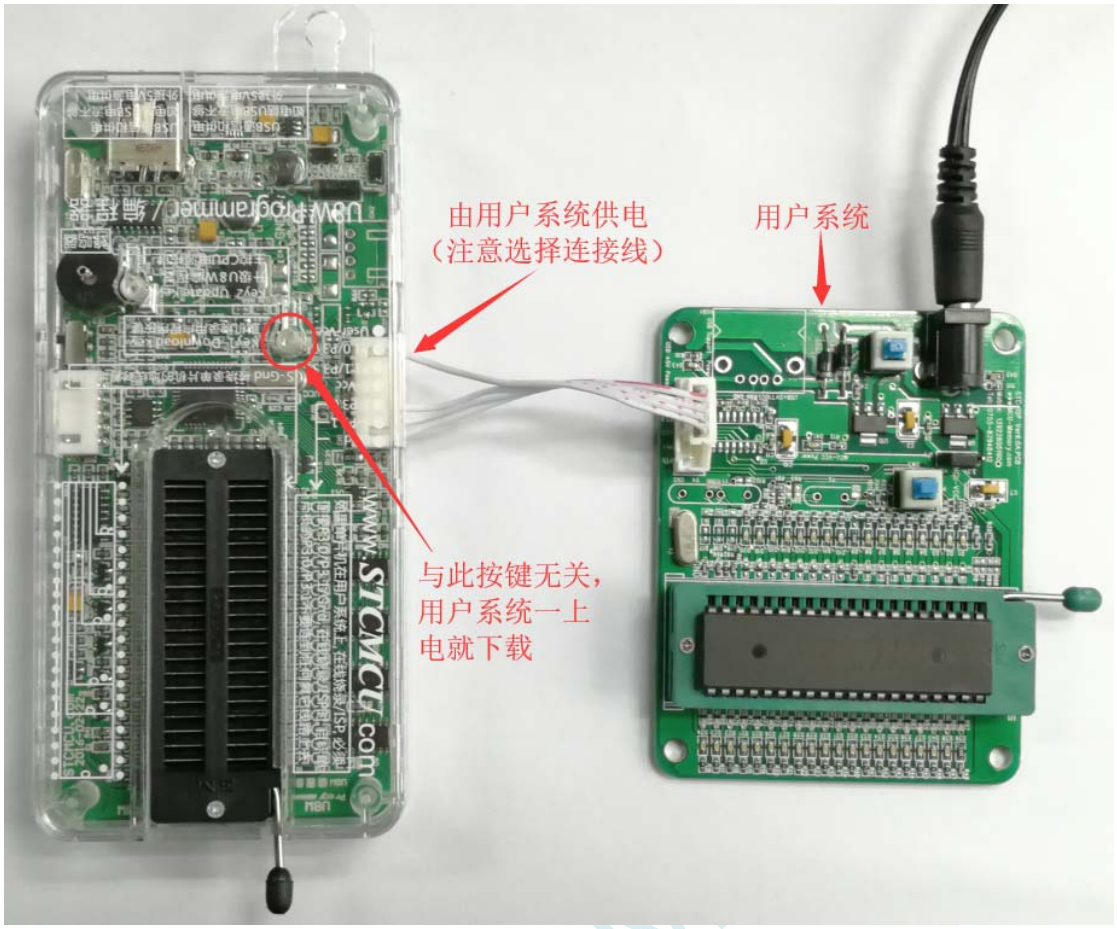
软件使用)。



1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8W 下载工具中。

（3）然后按下图所示的方式连接 U8W 与用户系统，给用户系统供电，即可开始脱机下载：



下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

目标芯片由用户系统引线连接 U8W 且 U8W 与用户系统各自独立供电进行脱机下载

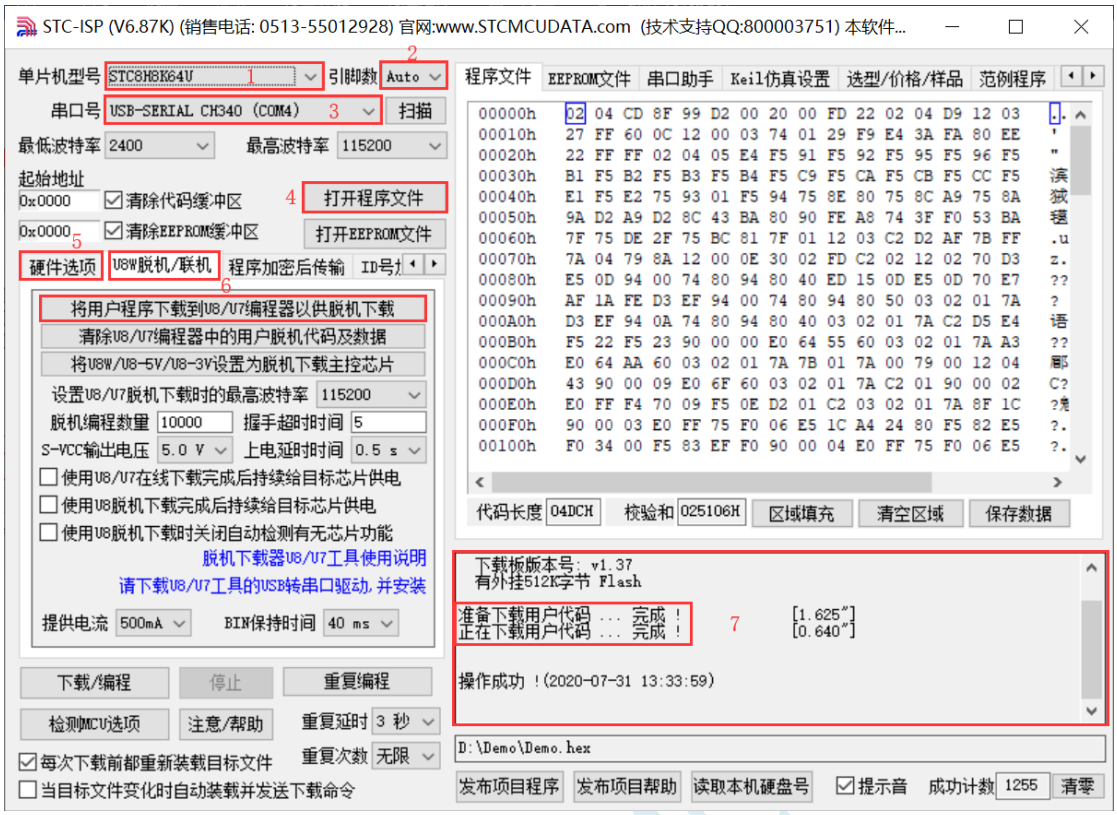
（1）首先使用 STC 提供的 USB 连接线将 U8W 下载板连接到电脑，如下图：



（2）在 STC-ISP 下载软件中按如下图所示的步骤进行设置：

建议用户用最新版本的 STC-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCMCUDATA.com 中 STC-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCMCUDATA.com 中下载最新版本的

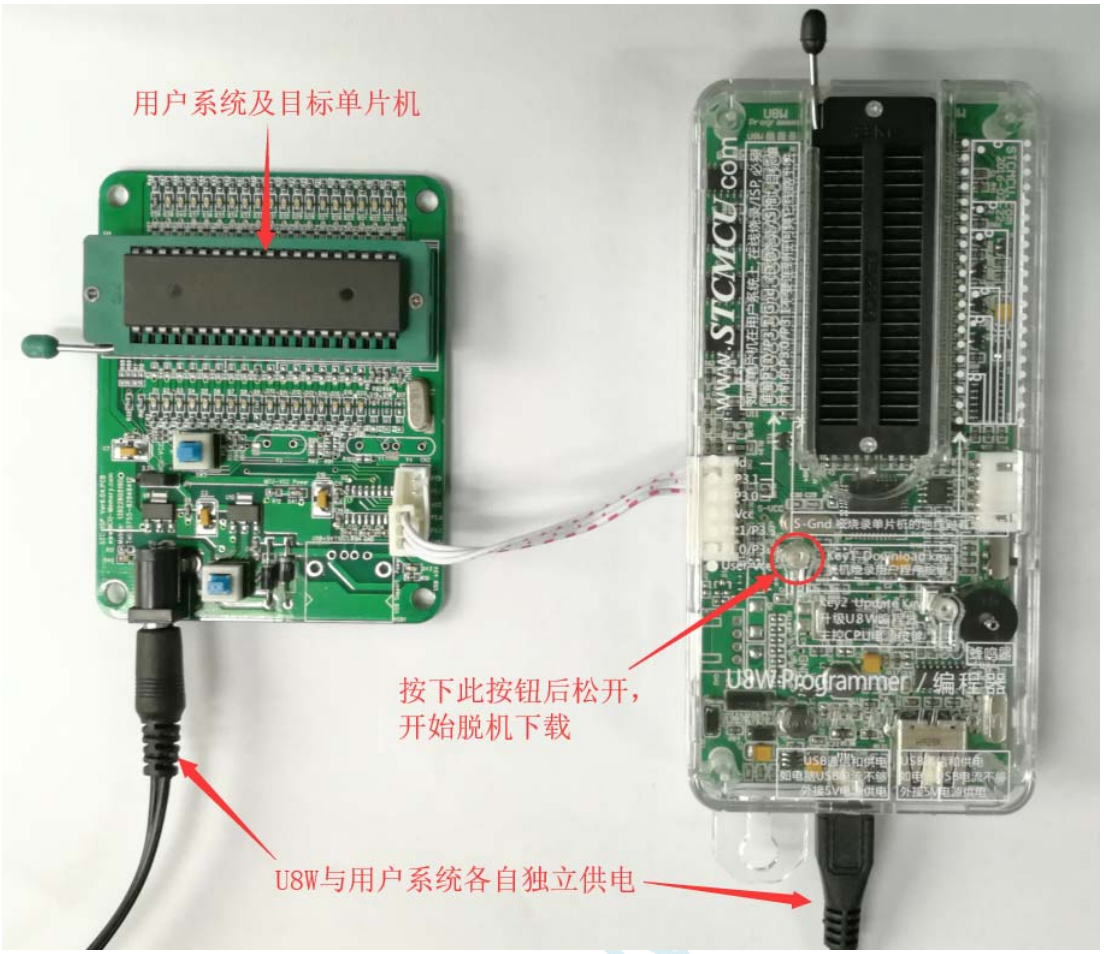
软件使用)。



1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8W 下载工具中。

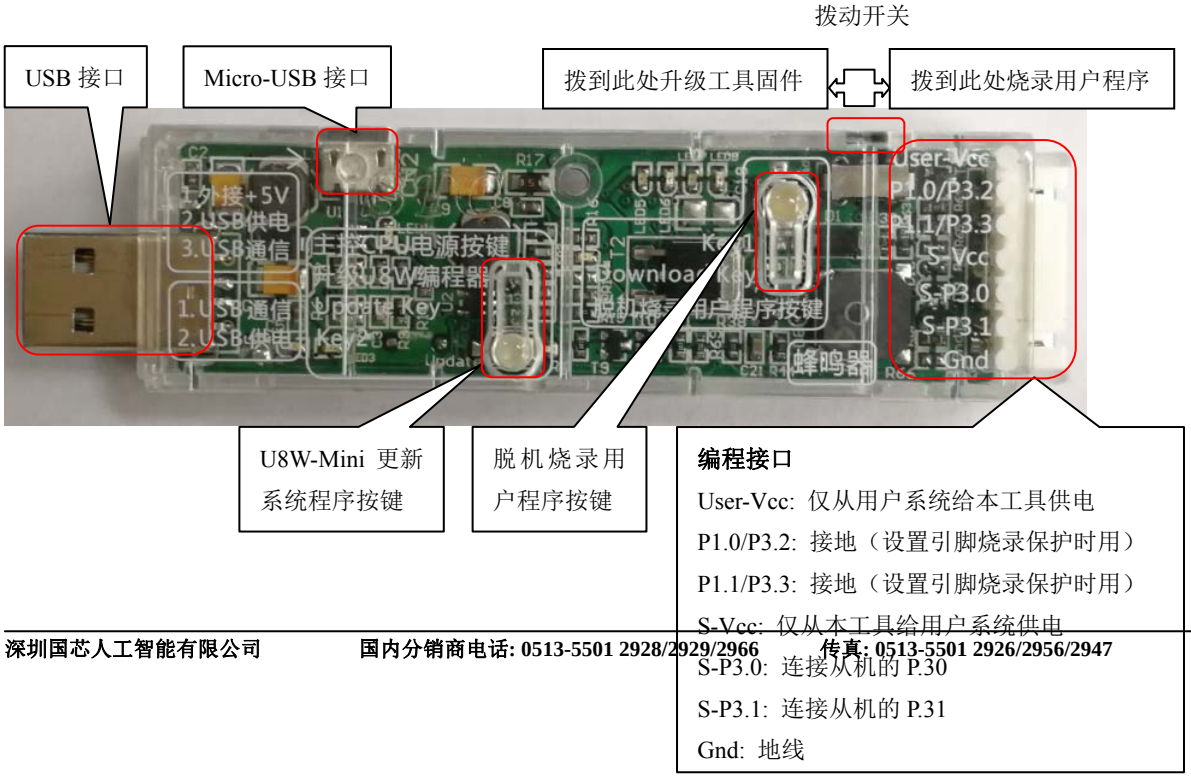
（3）然后按下图所示的方式连接 U8W 与用户系统，并将图中所示按钮先按下后松开，准备开始脱机下载，最后给用户系统上电/开电源，下载用户程序正式开始：



下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

F.3.5 U8W-Mini的功能介绍

下面详细介绍 U8W-Mini 工具的各主要接口及功能：



编程接口：根据不同的供电方式，使用不同的下载连接线连接 U8W-Mini 下载板和用户系统。

U8W-Mini 更新系统程序按键：用于更新 U8W-Mini 工具，当有新版本的 U8W 固件时，需要按下此按键对 U8W-Mini 的主控芯片进行更新（注意：必须先将更新/下载选择接口上的拨动开关拨动到升级工具固件）。

脱机下载用户程序按钮：开始脱机下载按钮。首先 PC 将脱机代码下载到 U8W-Mini 上，然后使用下载连接线将用户系统连接到 U8W-Mini，再按下此按钮即可开始脱机下载（每次上电时也会立即开始下载用户代码）。

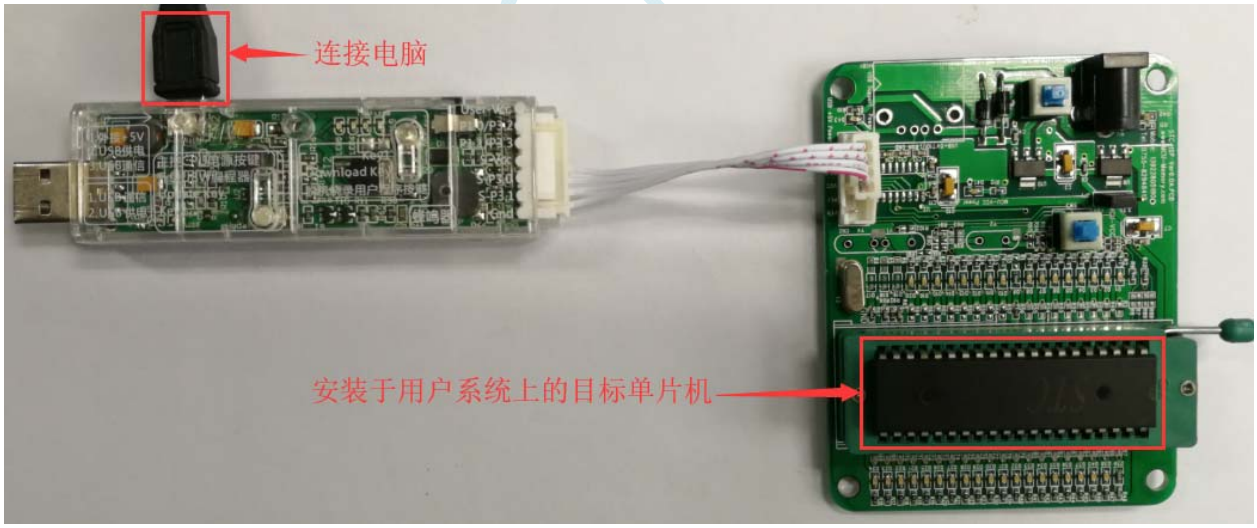
更新/下载选择接口：当需要对 U8W-Mini 的底层固件进行升级时，需将此拨动开关拨动到升级工具固件处，当需通过 U8W-Mini 对目标芯片进行烧录程序，则需将拨动开关拨动到烧录用户程序处。（拨动开关连接方式请参考上图）

USB 接口：USB 接口与 Micro-USB 接口是相同的功能，用户根据需要连接其中一个接口到电脑即可。

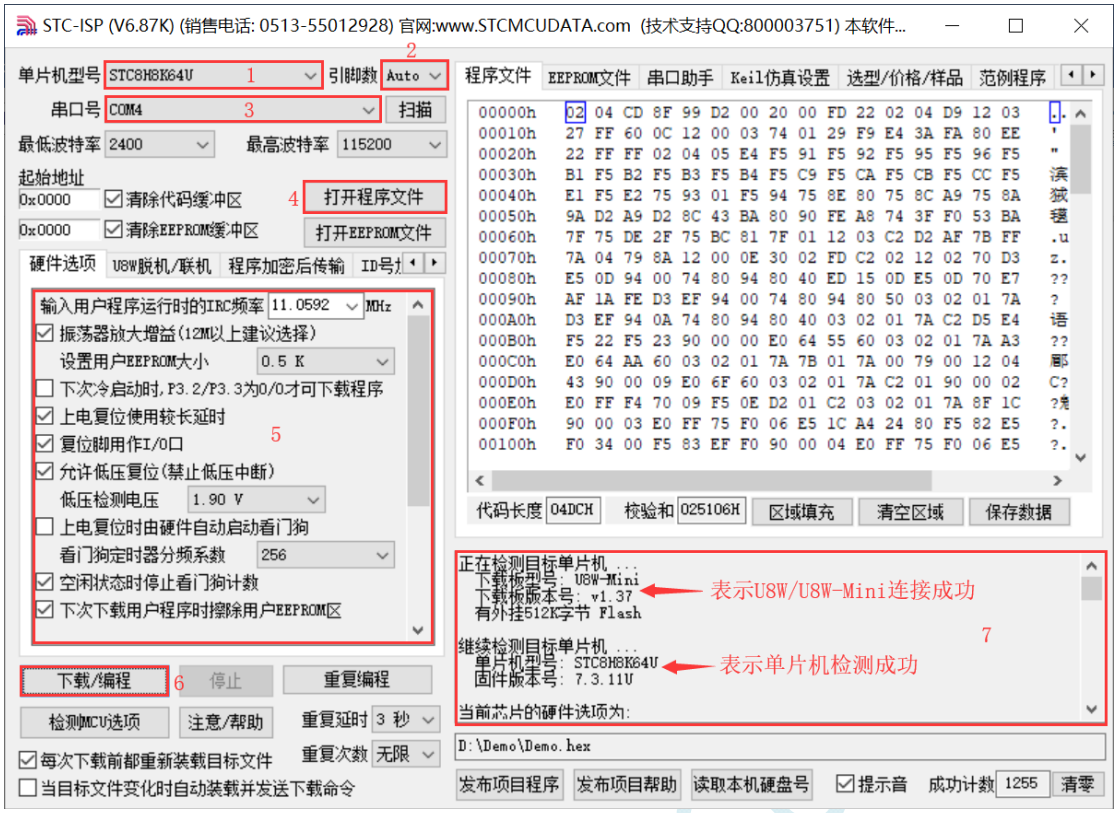
F.3.6 U8W-Mini的在线联机下载使用说明

目标芯片通过用户系统引线连接 U8W-Mini 并由 U8W-Mini 连接电脑进行在线联机下载

首先使用 STC 提供的 USB 连接线将 U8W-Mini 连接电脑，再将 U8W-Mini 通过下载线与用户系统的目标单片机相连接，连接方式如下图所示：



然后使用 STC-ISP 下载软件下载程序，步骤如下：



1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W-Mini 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W-Mini 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 点击“下载/编程”按钮开始烧录；
7. 显示烧录过程的步骤信息，烧录完成提示“操作成功！”。

当信息框中有输出下载板的版本号信息以及外挂 Flash 的相应信息时，表示已正确检测到 U8W-Mini 下载工具。

下载的过程中，U8W-Mini 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

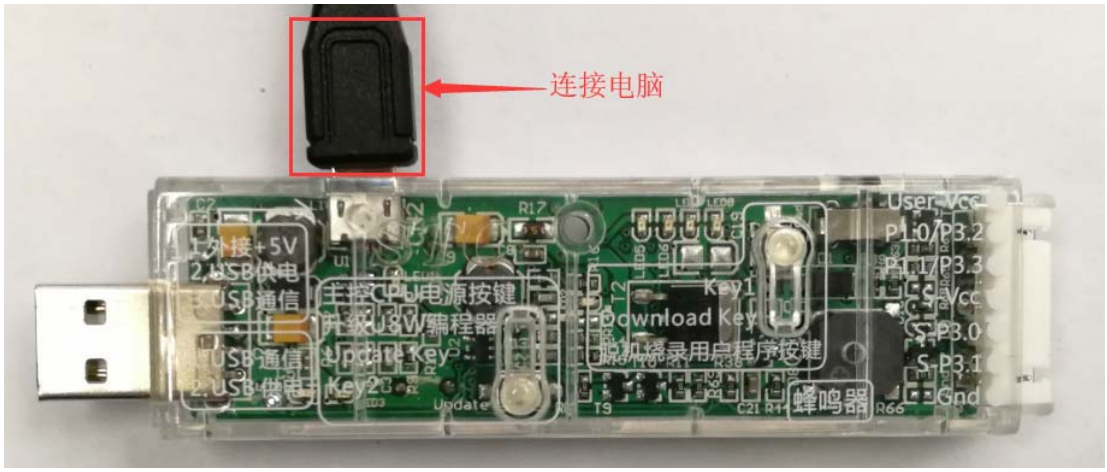
建议用户用最新版本的 STC-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCMCUDATA.com 中 STC-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCMCUDATA.com 中下载最新版本的软件使用）。

F.3.7 U8W-Mini的脱机下载使用说明

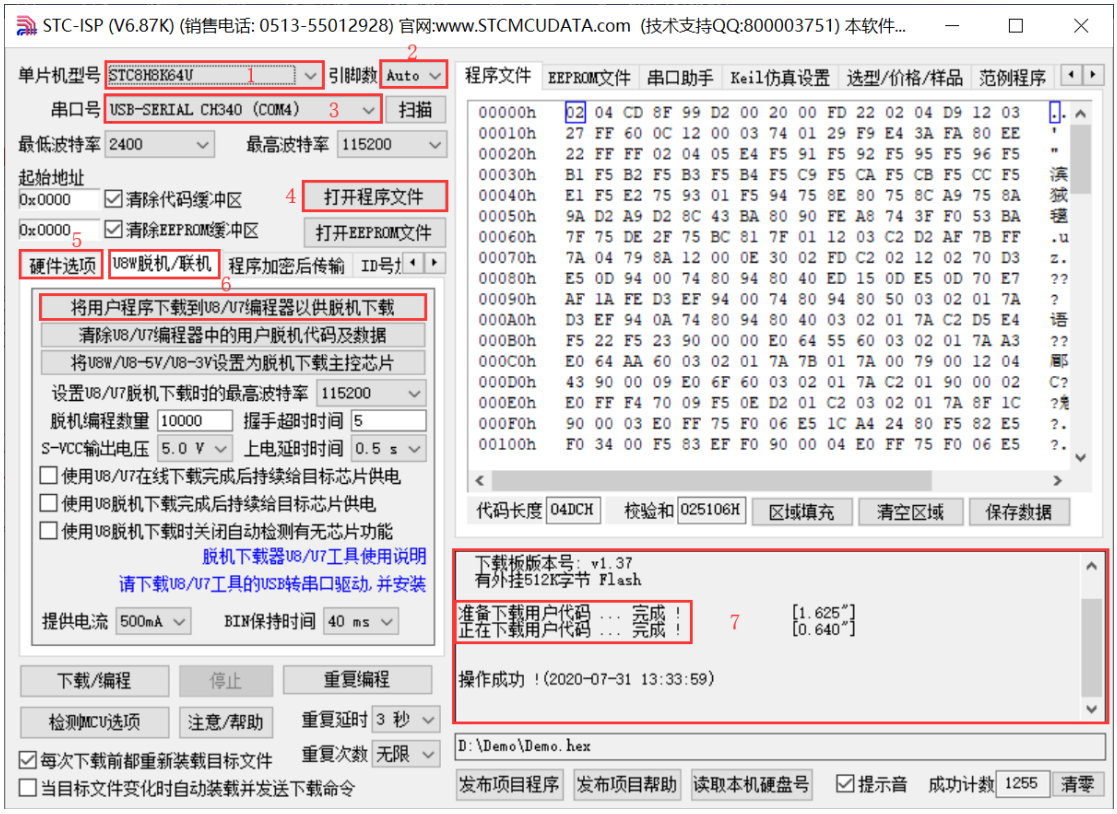
目标芯片由用户系统引线连接 U8W-Mini 并通过 USB 连接电脑给 U8W-Mini 供电进行脱机下载

使用 USB 给 U8W-Mini 供电从而进行脱机下载的步骤如下：

（1）使用 STC 提供的 USB 连接线将 U8W-Mini 下载板连接到电脑，如下图：



（2）在 STC-ISP 下载软件中按如下图所示的步骤进行设置：

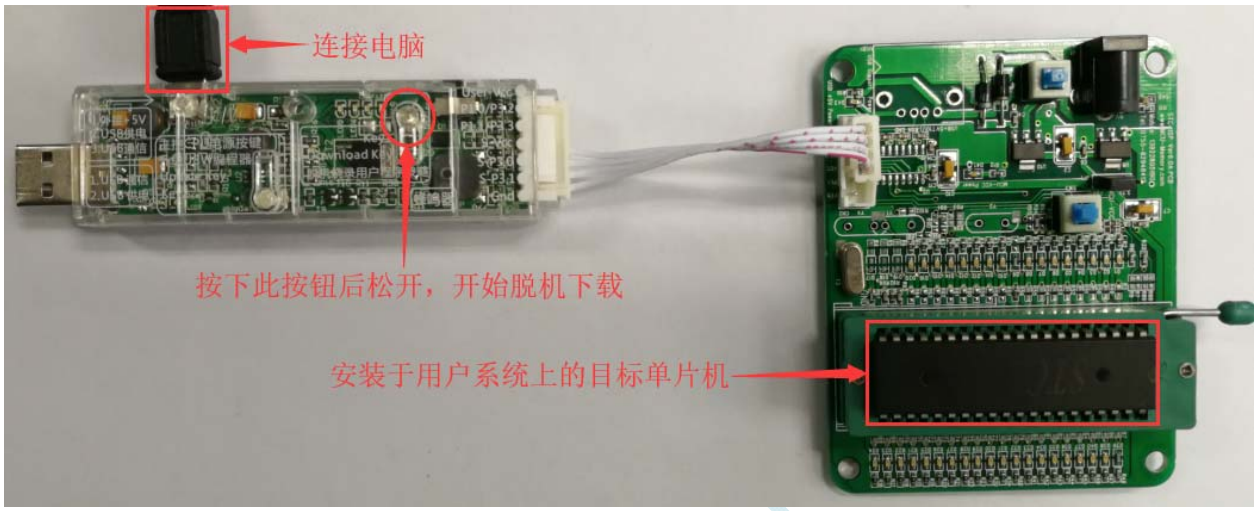


1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W-Mini 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W-Mini 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8W-Mini 下载工具中。

建议用户用最新版本的 STC-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCMCUDATA.com 中 STC-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCMCUDATA.com 中下载最新版本的软件使用）。

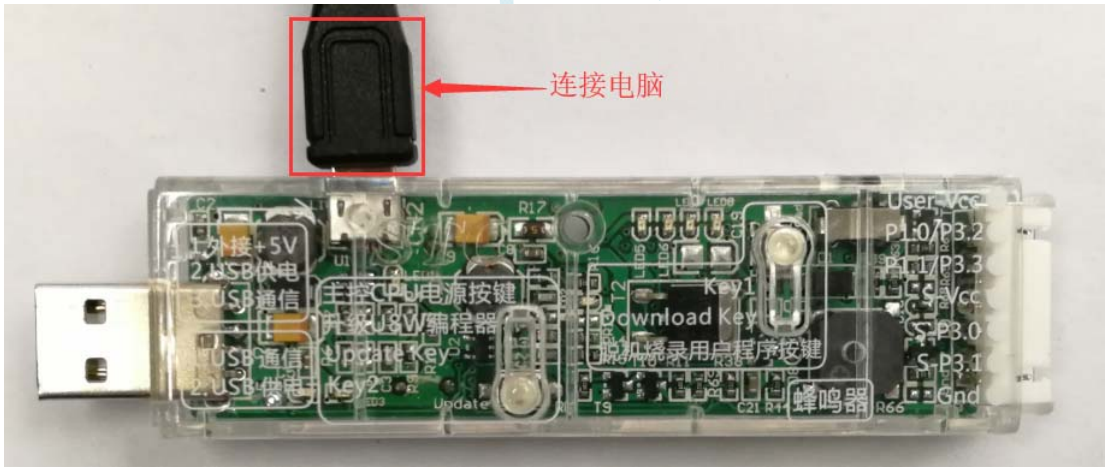
（3）然后使用连接线连接电脑、将 U8W-Mini 下载工具以及用户系统（目标单片机）如下图所示的方式连接起来，并按下图所示的按钮后松开，即可开始脱机下载：



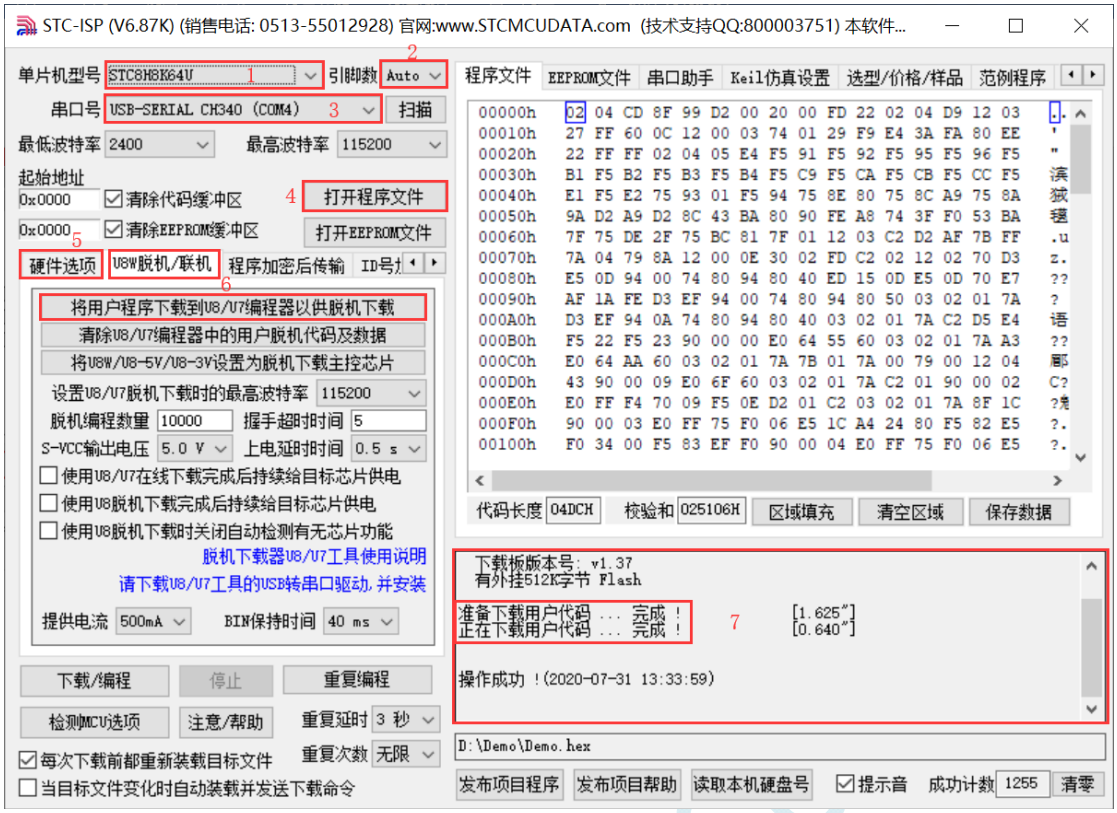
下载的过程中，U8W-Mini 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

目标芯片由用户系统引线连接 U8W-Mini 并通过用户系统给 U8W-Mini 供电进行脱机下载

（1）首先使用 STC 提供的 USB 连接线将 U8W-Mini 下载板连接到电脑，如下图：



（2）在 STC-ISP 下载软件中按如下图所示的步骤进行设置：

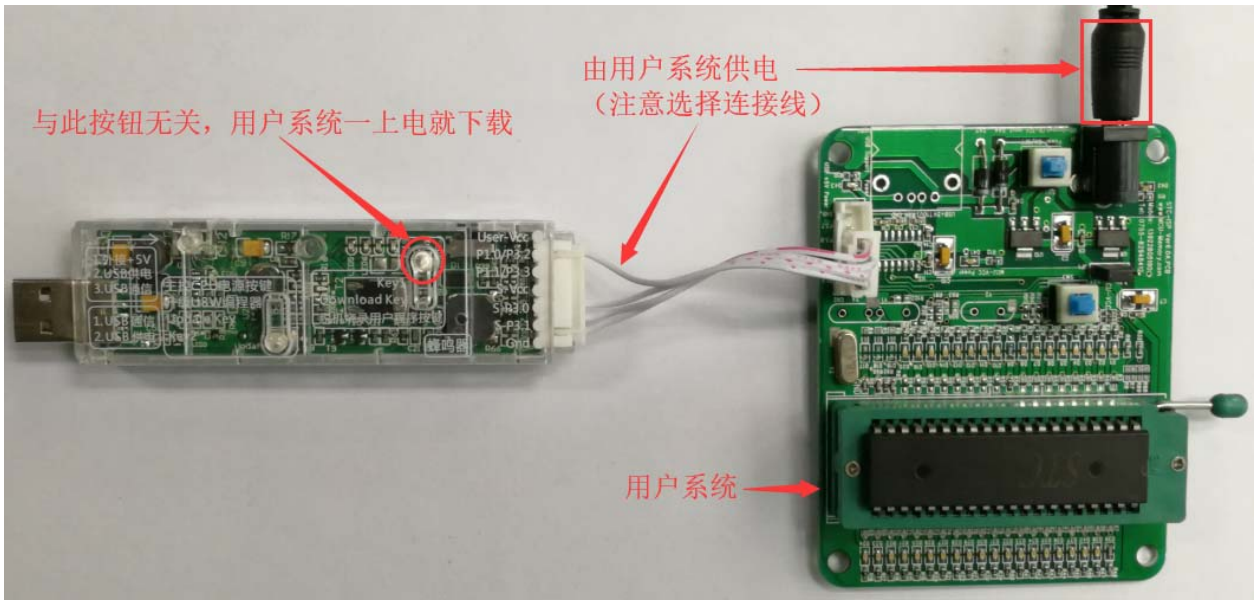


1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W-Mini 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W-Mini 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8W-Mini 下载工具中。

建议用户用最新版本的 STC-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCMCUDATA.com 中 STC-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCMCUDATA.com 中下载最新版本的软件使用）。

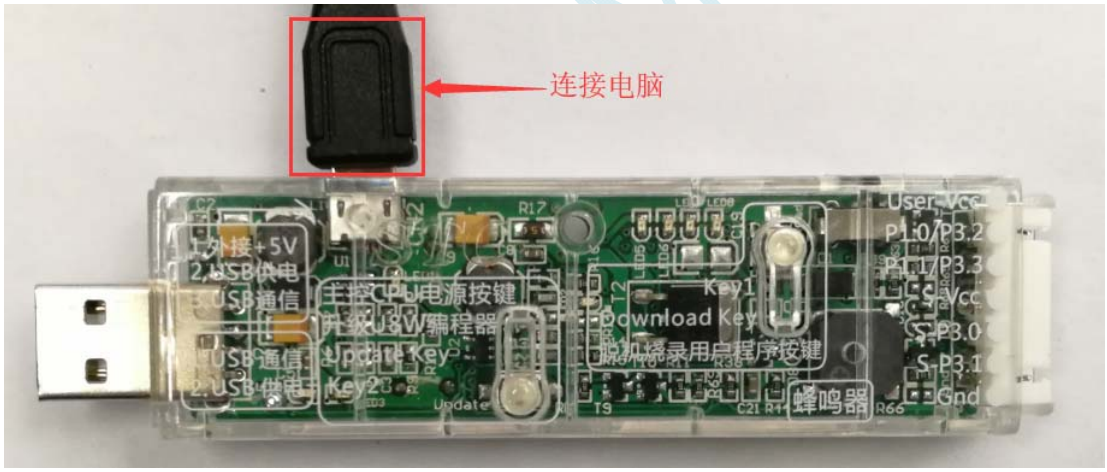
（3）然后按下图所示的方式连接 U8W-Mini 与用户系统，用户系统一上电就开始脱机下载：



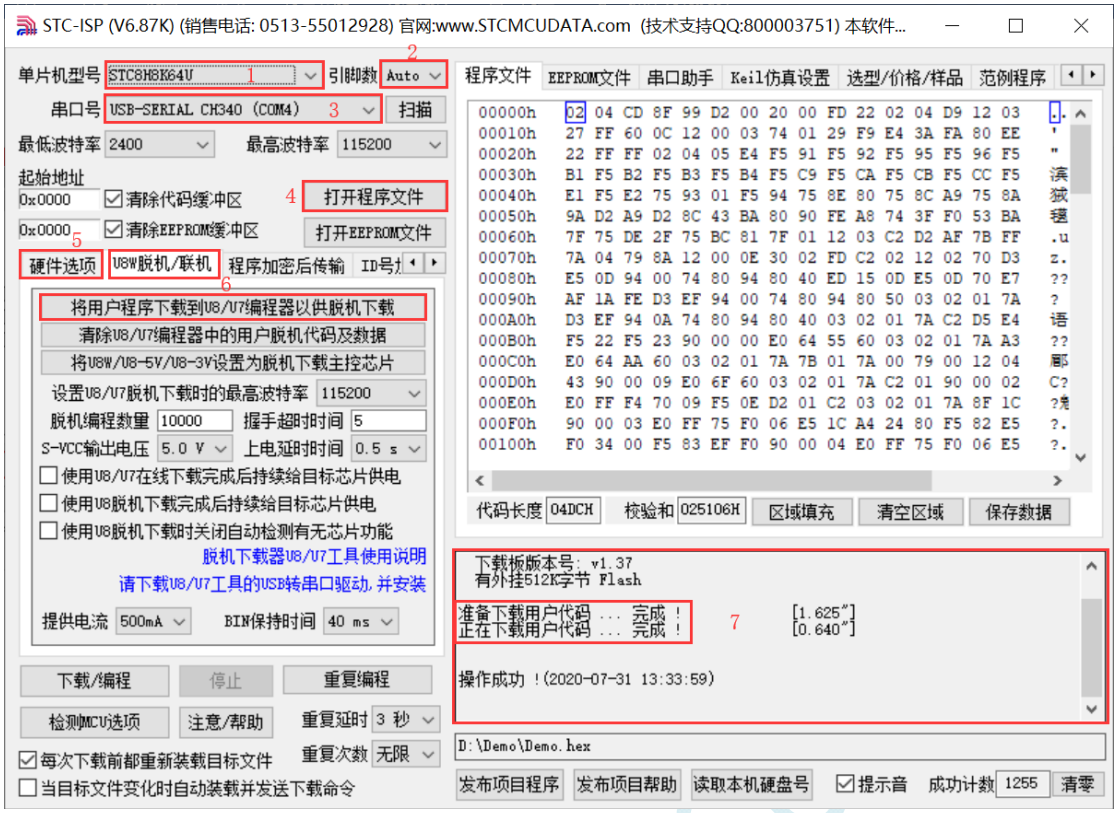
下载的过程中，U8W-Mini 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

目标芯片由用户系统引线连接 U8W-Mini 且 U8W-Mini 与用户系统各自独立供电进行脱机下载

（1）首先使用 STC 提供的 USB 连接线将 U8W-Mini 下载板连接到电脑，如下图：



（2）在 STC-ISP 下载软件中按如下图所示的步骤进行设置：

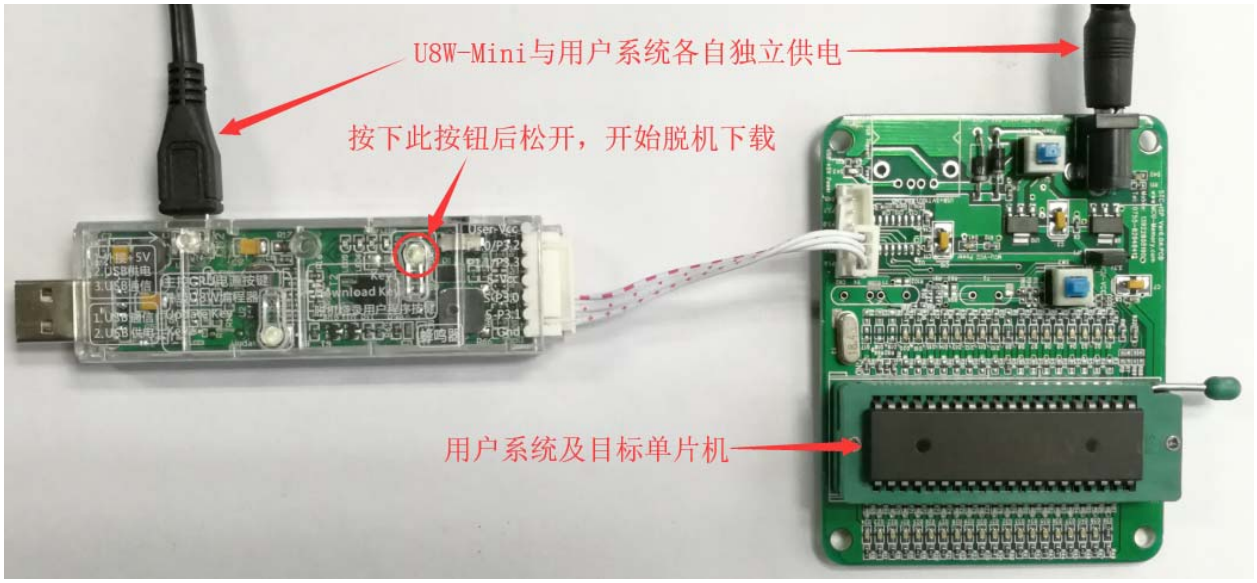


1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W-Mini 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W-Mini 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8W-Mini 下载工具中。

建议用户用最新版本的 STC-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCMCUDATA.com 中 STC-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCMCUDATA.com 中下载最新版本的软件使用）。

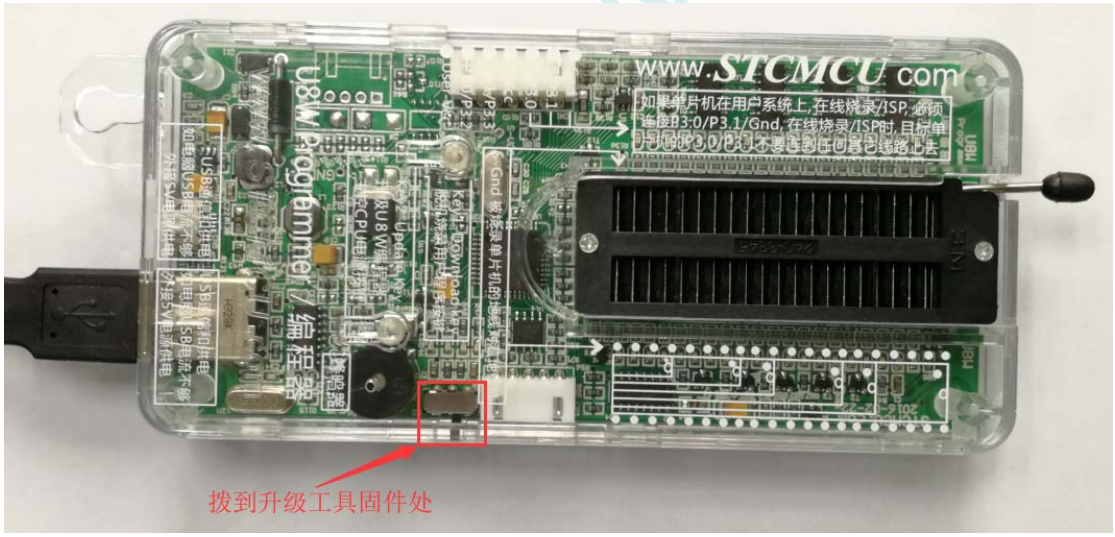
（3）然后按下图所示的方式连接 U8W-Mini 与用户系统，并将图中所示按钮先按下后松开，准备开始脱机下载，最后给用户系统上电/开电源，下载用户程序正式开始：



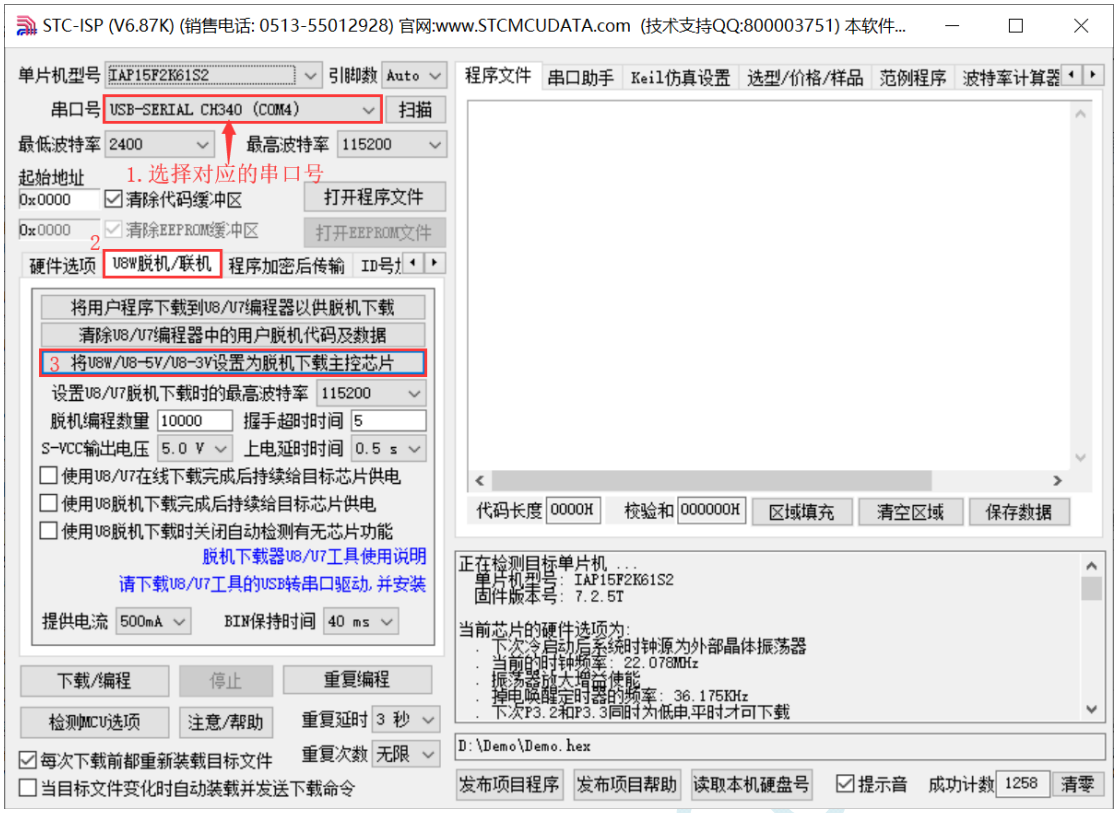
下载的过程中，U8W-Mini 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

F.3.8 制作/更新U8W/U8W-Mini

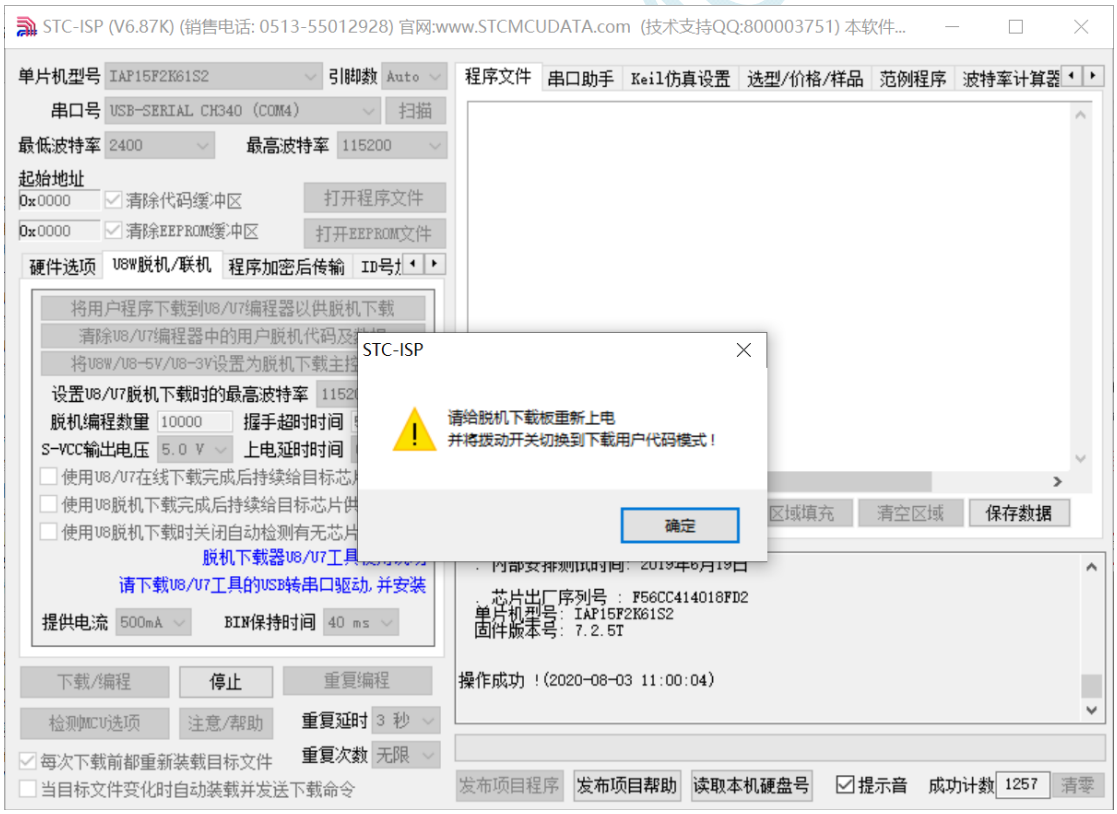
制作 U8W/U8W-Mini 下载母片的过程类似，为节约篇幅，下文以 U8W 为例，详述如何制作 U8W 下载母片。在制作 U8W 下载母片之前需要将 U8W 下载板的“更新/下载选择接口”拨到“升级工具固件”，如下图所示：



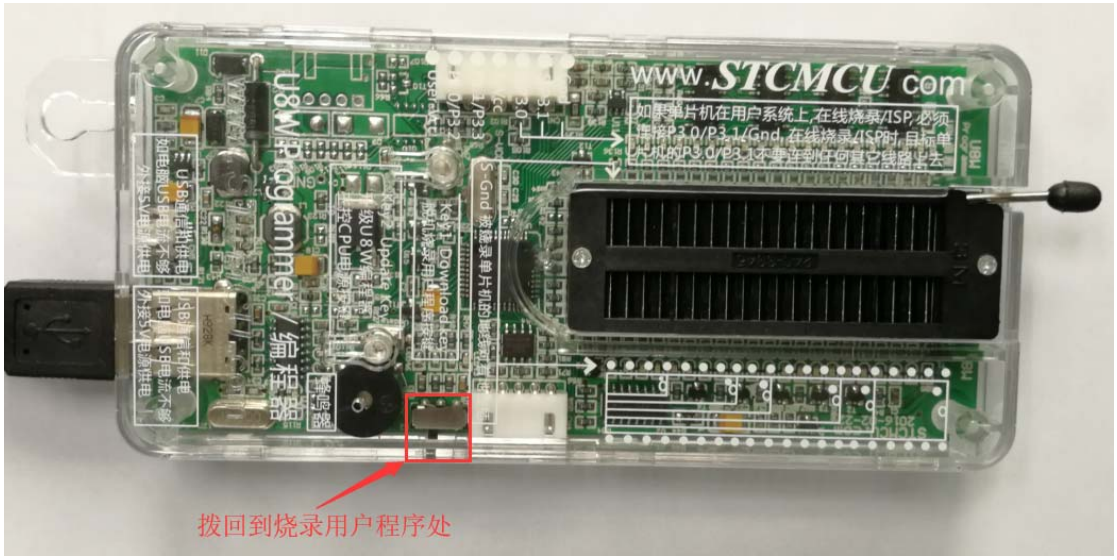
然后在 STC-ISP 下载程序中的“U8W 脱机/联机”页面中点击“将 U8W/U8-5V/U8-3V 设置为脱机下载主控芯片”按钮，如下图：（注意：一定要选择 U8W 所对应的串口）



在出现如下画面表示 U8W 控制芯片制作完成：



制作完成后，一定不要忘记将 U8W 的“更新/下载选择接口”拨回到“烧录用户程序”模式，并将 U8W 下载工具重新上电，如下图所示：（否则将不能正常进行烧录）



F.3.9 U8W/U8W-Mini设置直通模式（可用于仿真）

若要使用 U8W/U8-Mini 进行仿真，首先必须将 U8W/U8-Mini 设置为直通模式。U8W/U8W-Mini 实现 USB 转串口直通模式的方法如下：

1. 首先 U8W/U8W-Mini 固件必须升级到 v1.37 及以上版本；
2. U8W/U8W-Mini 上电后为正常下载模式，此时按住工具上的 Key1（下载）按键不要松开，再按一下 Key2（电源）按键，然后放开 Key2（电源）按键 后，再松开 Key1（下载）按键，U8W/U8W-Mini 会进入 USB 转串口直通模式。（按下 Key1 ➔ 按下 Key2 ➔ 松开 Key2 ➔ 松开 Key1）；
3. 进入直通模式的 U8W/U8W-Mini 工具只是简单的 USB 转串口不具备脱机下载功能，若需要恢复 U8W/U8W-Mini 的原有功能，只需要再次单独按一下 Key2（电源）按键即可。

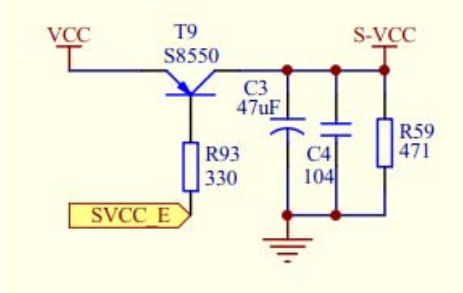
F.3.10 U8W/U8W-Mini的参考电路

USB 型联机/脱机下载板 U8W/U8W-Mini 为用户提供了如下的常用控制接口：

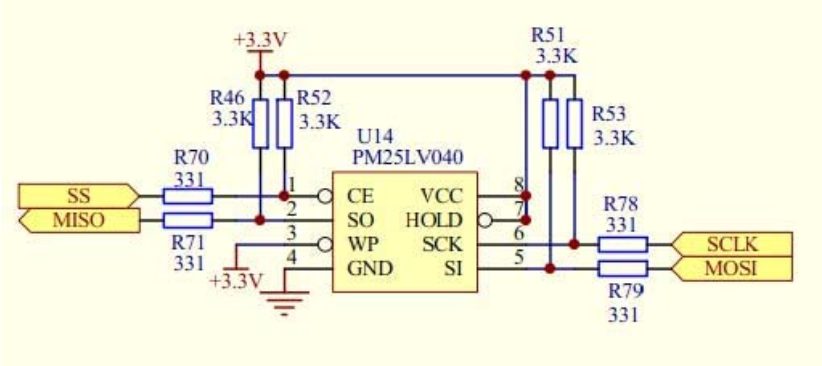
脚位功能	端口	功能描述
电源控制脚	P2.6	低位有效
下载通讯脚	P1.0	串口 RXD，连接目标芯片的 TXD（P3.1）
	P1.1	串口 TXD，连接目标芯片的 RXD（P3.0）
编程按键	P3.6	低有效
显示	P3.2	LED1
	P3.3	LED2
	P3.4	LED3
	P5.5	LED4
外挂串行 Flash 控制脚	P2.4	Flash 的 CE 脚
	P2.2	Flash 的 SO 脚
	P2.3	Flash 的 SI 脚
	P2.1	Flash 的 SCLK 脚

全自动烧录工具 分选机信号	P3.6	起始信号
	P1.5	完成信号
	P5.4	OK 信号（良品信号）
	P3.7	ERROR 信号（不良品信号）
蜂鸣器（BEEP）控制	P2.5	高有效（高电平发出声音）

电源控制部分参考电路图：

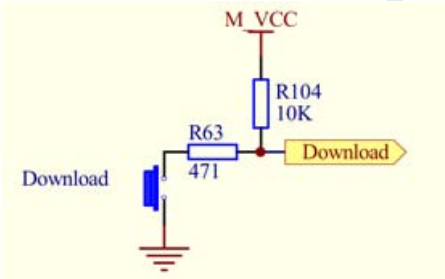


Flash 控制部分参考电路图：

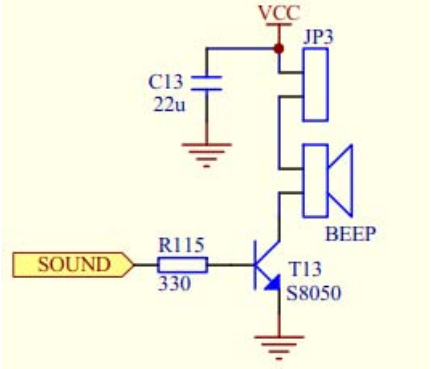


用户程序大于 41K 时需要此 Flash 存储器

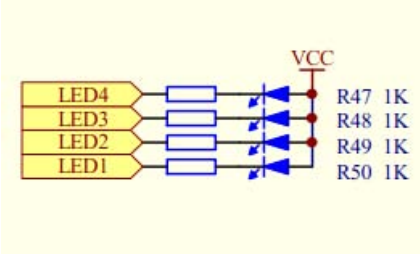
按键部分参考电路图：



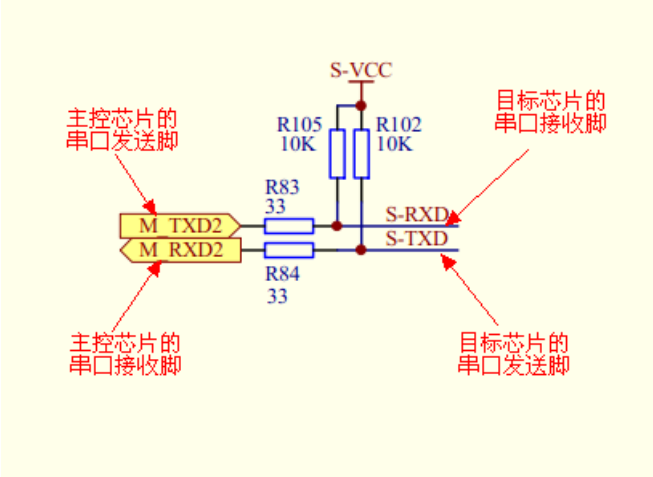
蜂鸣器部分参考电路图：



LED 显示部分参考电路图：



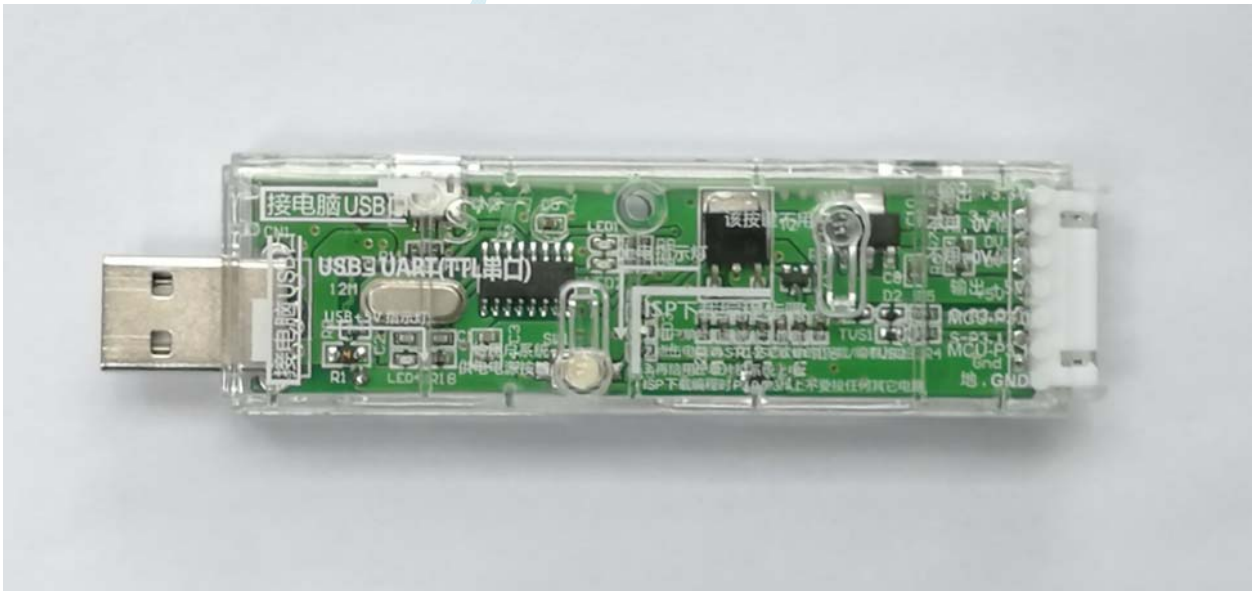
串口通讯脚连接部分参考电路图：



F.4 STC通用USB转串口工具

F.4.1 STC通用USB转串口工具外观图

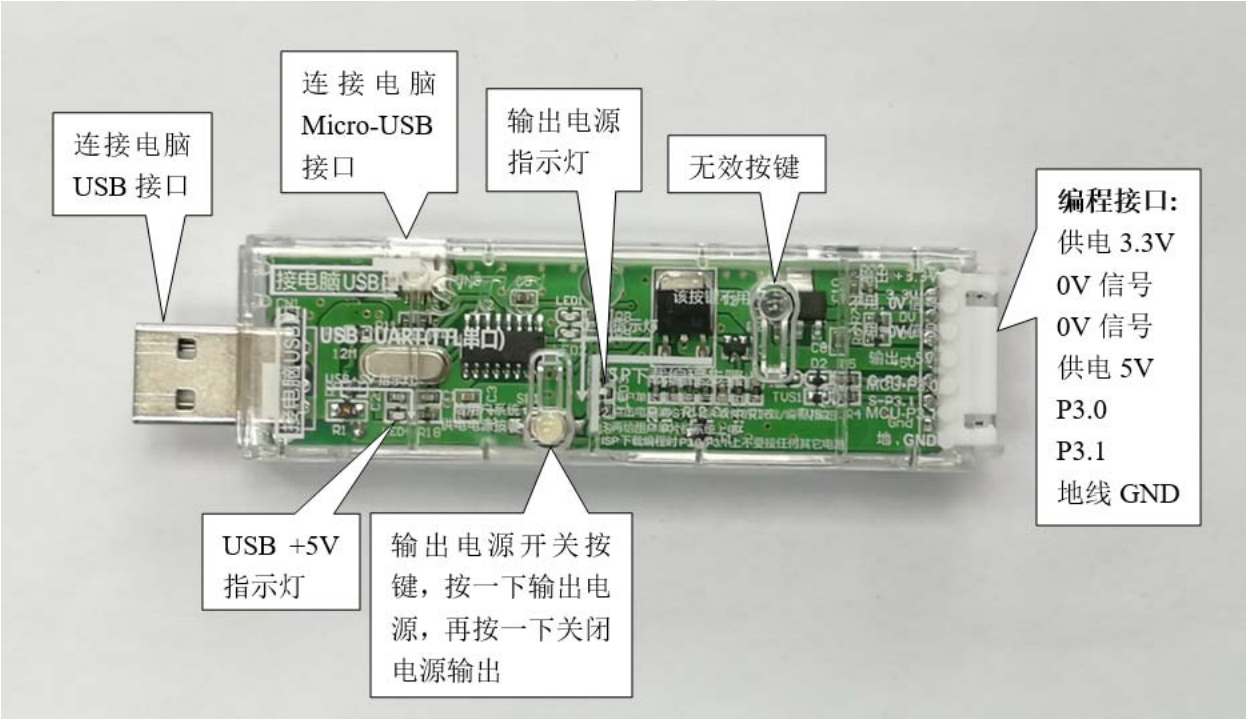
正面：



背面：



F.4.2 STC通用USB转串口工具布局图



在此，需要对“电源开关”进行说明：

此按钮的作用与自锁开关相同，在开关按钮第一次按时，开关接通电源并保持，即自锁，在开关按钮第二次按时，开关断开电源。鉴于自锁开关使用过程中容易损坏的特点，我们设计了一套利用轻触开关替代自锁开关功能的电路，提高工具的使用寿命。

而对于 STC 的单片机，要想进行 ISP 下载，则必须是在上电复位时接收到串口命令才会开始执行 ISP

程序，所以使用 STC 通用 USB 转串口工具下载程序到 MCU 的正确步骤为：

1. 使用 STC 通用 USB 转串口工具将待烧录 MCU 与电脑进行连接；
2. 打开 STC 的 ISP 下载软件；
3. 选择单片机型号；
4. 选择 STC 通用 USB 转串口工具所对应的串口；
5. 打开目标文件（HEX 格式或者 BIN 格式）；
6. 点击 ISP 下载软件中的“下载/编程”按钮；
7. 按一下 STC 通用 USB 转串口工具上的“电源开关”给 MCU 供电，即可开始下载。
【冷启动烧录】

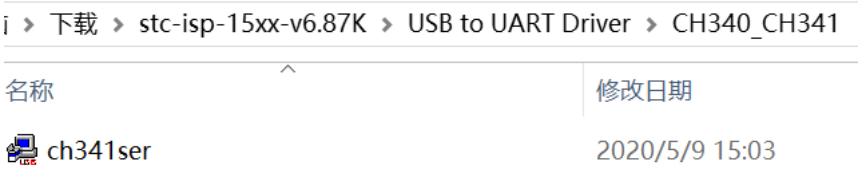
此外，USB 接口与 Micro-USB 接口是相同的功能，用户根据需要连接其中一个接口到电脑即可。
编程接口的 0V 信号脚内部有 470 欧姆电阻接地，如果设置了 P1.0/P1.1=0/0 或者 P3.2/P3.3=0/0 时才能下载，可将 P1.0，P1.1 或者 P3.2，P3.3 接到 0V 信号脚。

F.4.3 STC通用USB转串口工具驱动安装

STC 通用 USB 转串口工具采用 CH340 USB 转串口芯片(可以外挂晶振,更精准),只要下载通用的 CH340 串口驱动程序进行安装即可，以下是 STC 官网（www.STCMCUDATA.com）提供的 CH341SER 串口驱动下载位置：



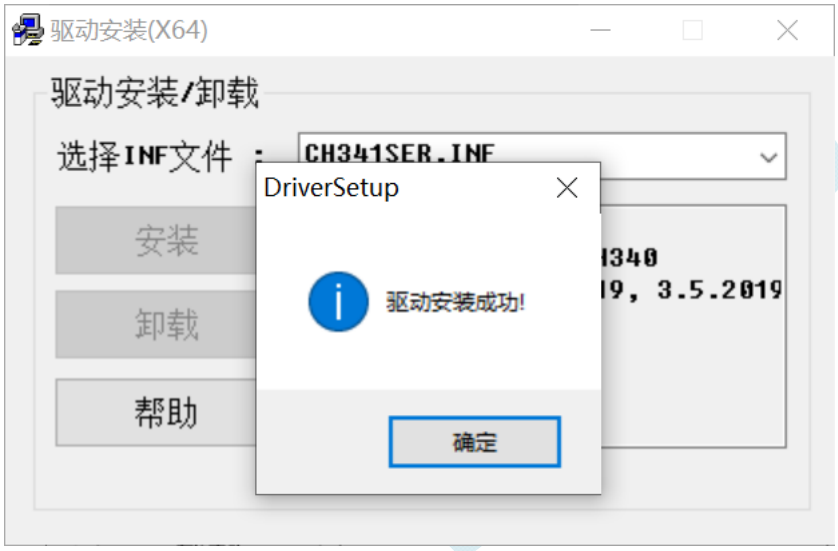
下载后进行解压，CH340 的驱动安装包路径 stc-isp-15xx-v6.87K\USB to UART Driver\CH340_CH341：



以 STC 官网提供的 CH341SER 串口驱动为例，双击“CH341SER.exe”安装包，在弹出的主界面点击“安装”按钮开始安装驱动程序：

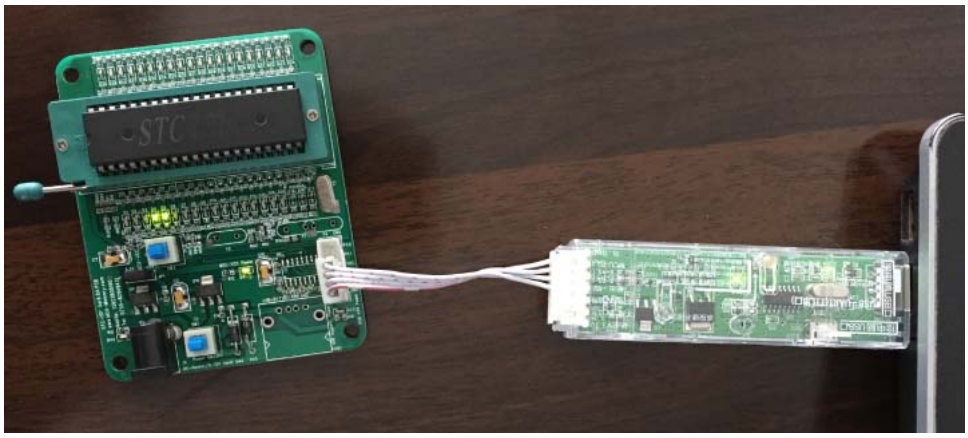


然后弹出驱动安装成功对话框，点击“确定”按钮完成安装：

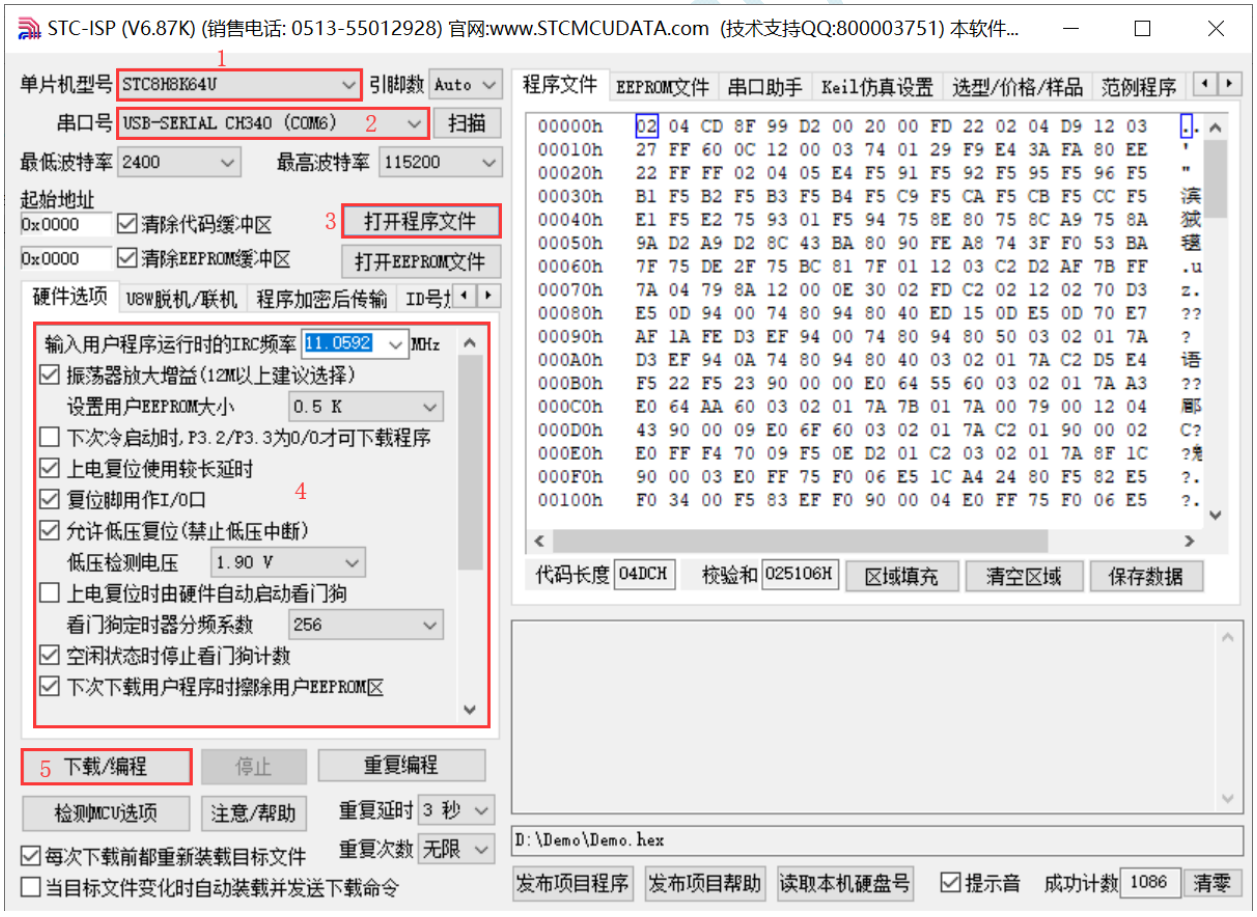


F.4.4 使用STC通用USB转串口工具下载程序到MCU

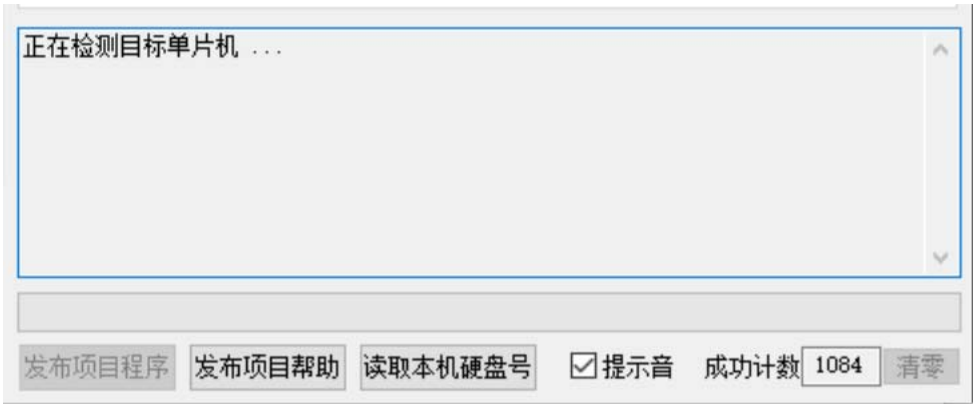
1. 使用 STC 通用 USB 转串口工具将待烧录 MCU 与电脑进行连接：



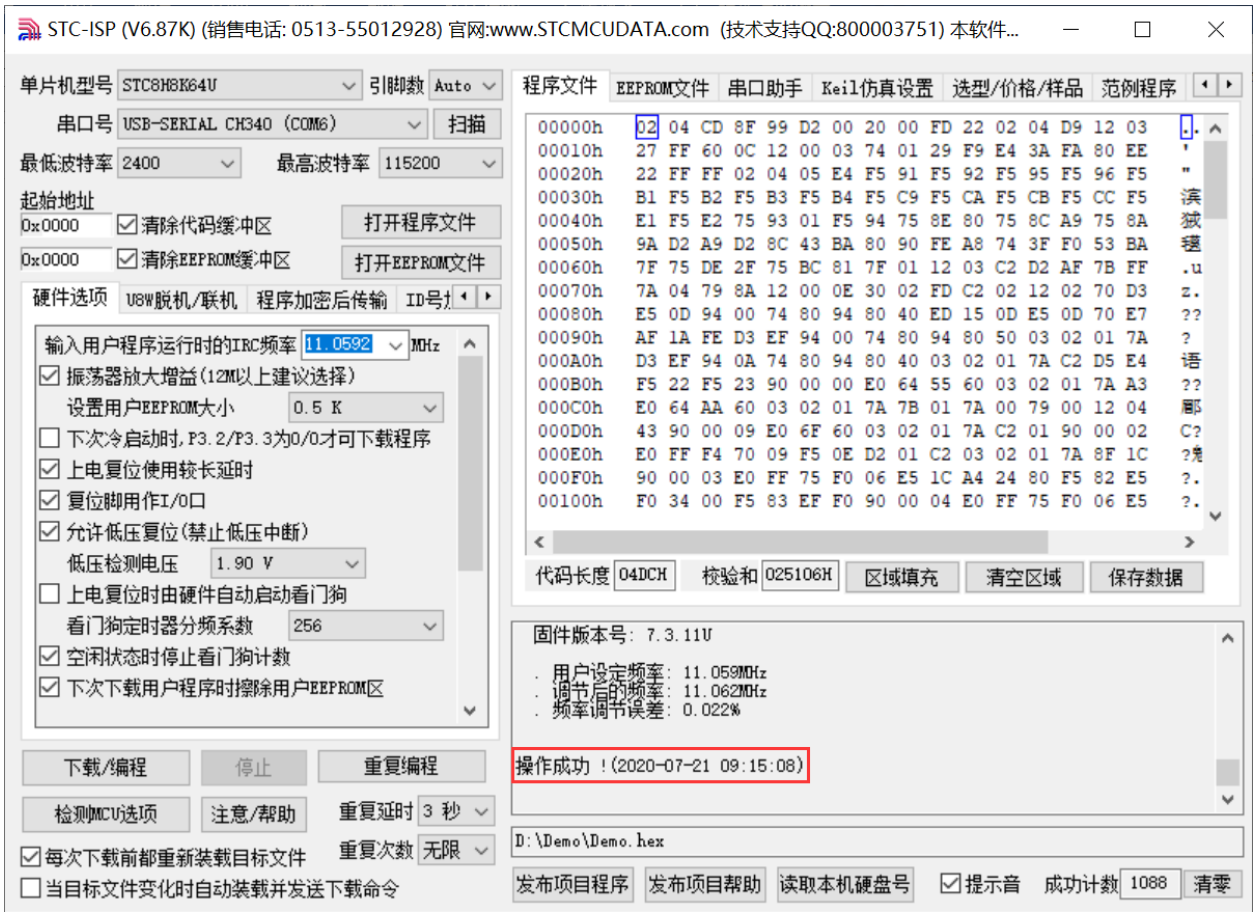
2. 打开 STC-ISP 软件；
3. 选择烧录芯片对应的型号；
4. 选择 STC 通用 USB 转串口工具所识别的串口号（当 STC 通用 USB 转串口工具与电脑正确连接后，软件会自动扫描并识别名称为“USB-SERIAL CH340 (COMx)”串口，具体的 COM 编号会因电脑不同而不同）。当有多个 USB 转串口线与电脑相连时，则必须手动选择；
5. 加载烧录程序；
6. 设置烧录选项；
7. 点击“下载/编程”按钮；



8. 右下角提示框显示“正在检测目标单片机 ...”时按一下 STC 通用 USB 转串口工具上的“电源开关”给 MCU 供电，即可开始下载【冷启动烧录】；



9. 等待下载结束，若下载成功，右下角提示框会显示“操作成功!”。

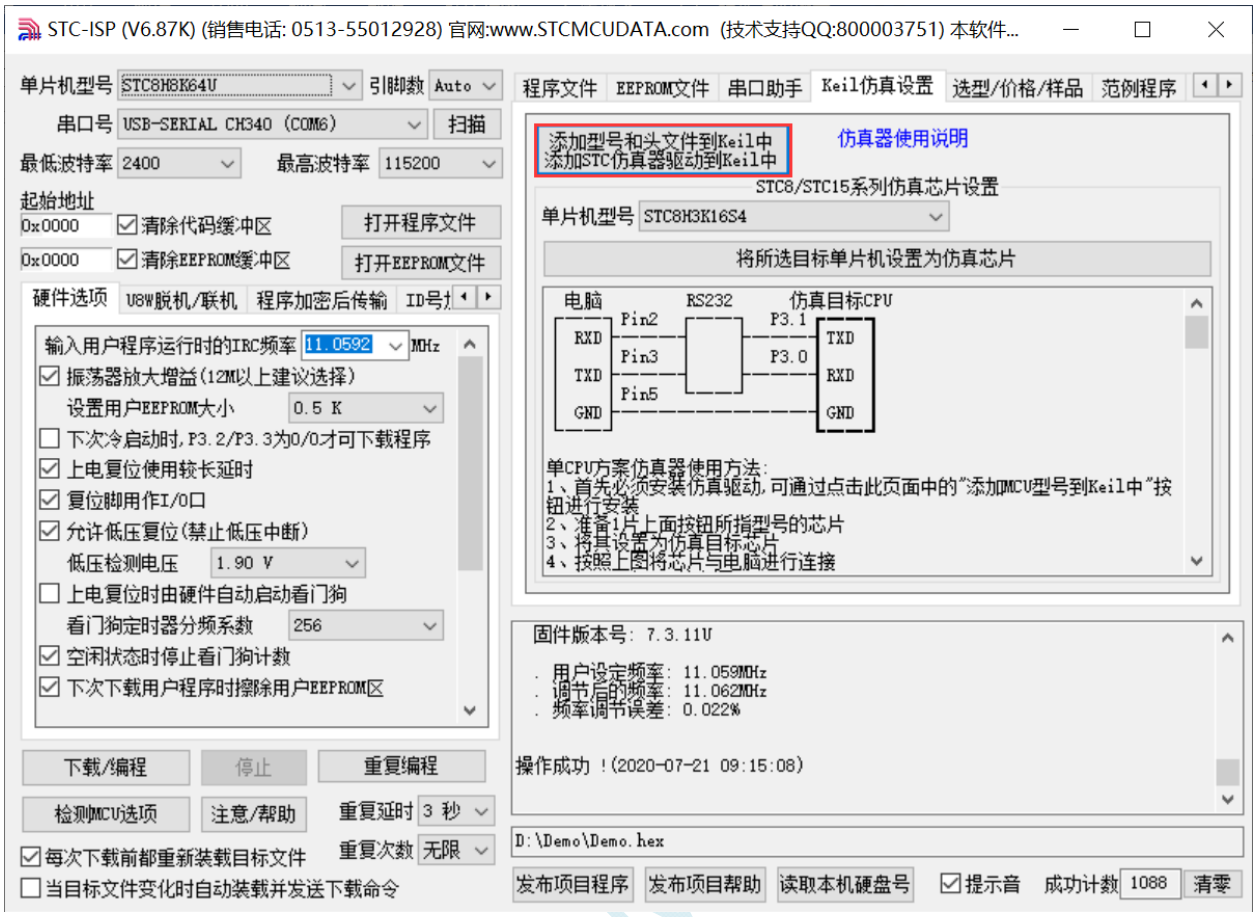


F.4.5 使用STC通用USB转串口工具仿真用户代码

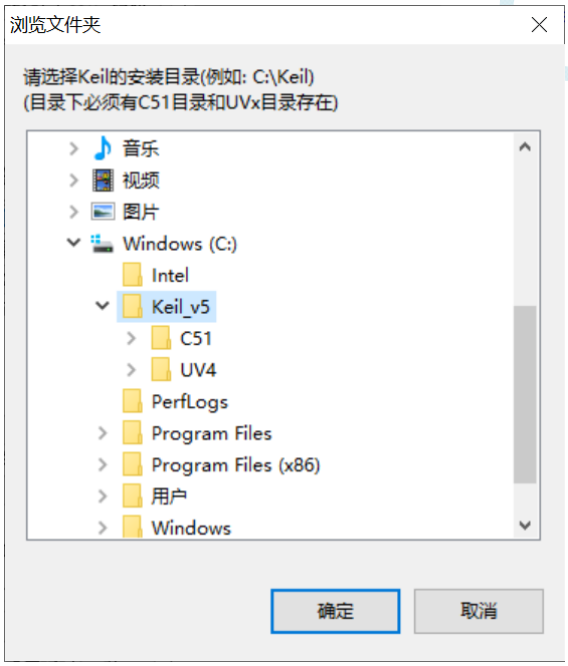
目前 STC 的仿真都是基于 Keil 环境的，所以若需要使用 STC 通用 USB 转串口工具仿真用户代码，则必须要安装 Keil 软件。

Keil 软件安装完成后，还需要安装 STC 的仿真驱动。STC 的仿真驱动的安装步骤如下：
首先开 STC-ISP 下载软件；
然后在软件右边功能区的“Keil 仿真设置”页面中点击“添加型号和头文件到 Keil 中 添加 STC 仿真器

驱动到 Keil 中”按钮：



按下后会出现如下画面：

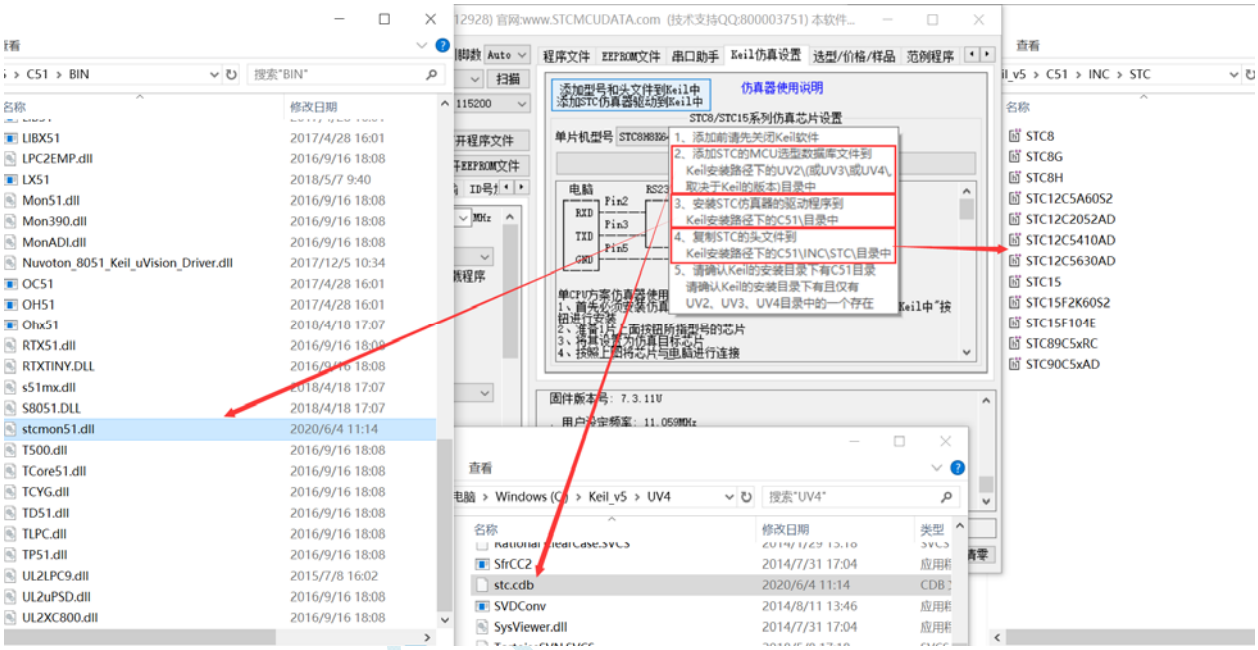


将目录定位到 Keil 软件的安装目录，然后确定。

安装成功后会弹出如下的提示框：

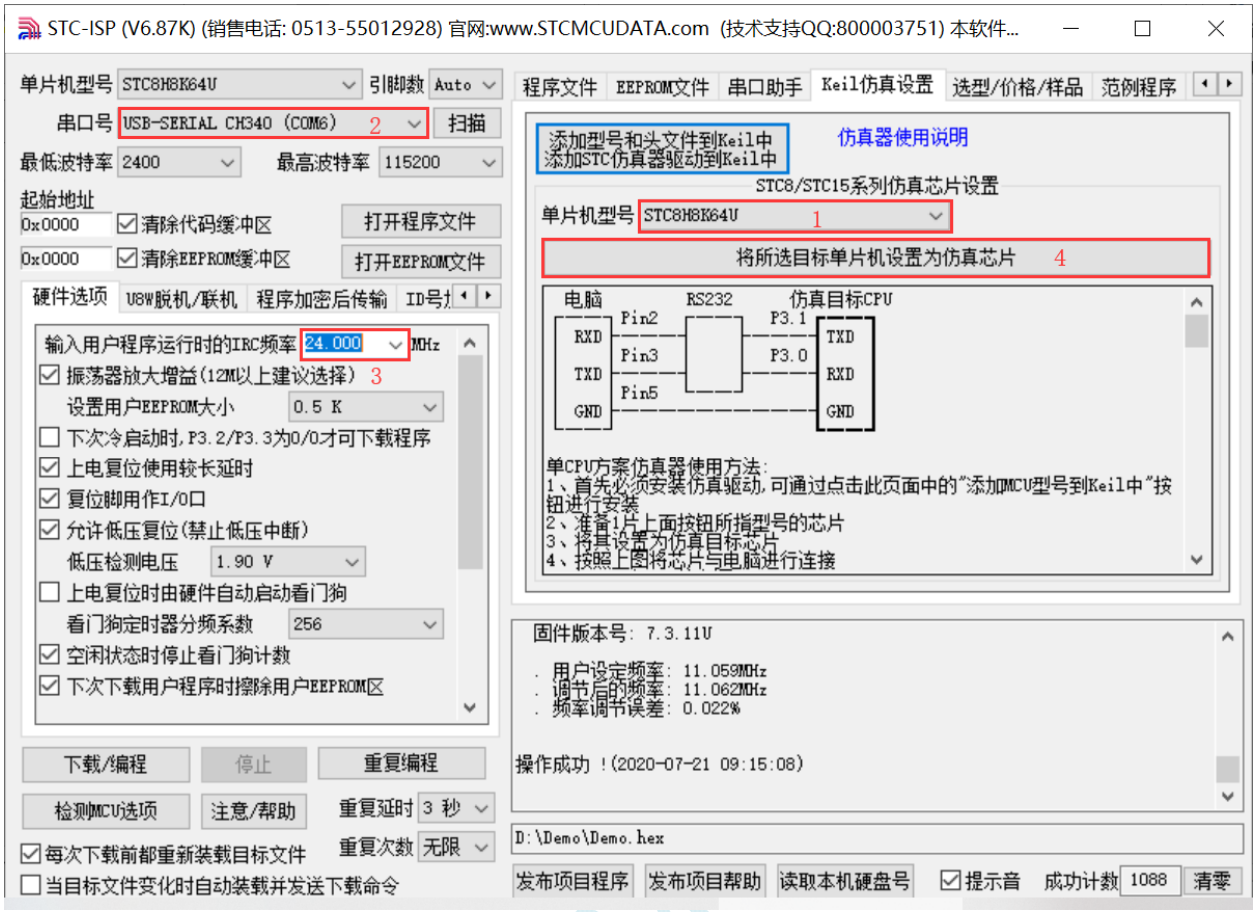


在 Keil 的相关目录中可以看到如下的文件，即表示驱动正确安装了。



由于在默认状态下，STC 的主控芯片并不是一颗仿真芯片，不具有仿真功能，所以若需要进行仿真，则还需要将 STC 的主控芯片设置为仿真芯片。
制作仿真芯片步骤如下：

首先使用 STC 通用 USB 转串口工具将 MCU 与电脑进行连接；
然后打开 STC 的 ISP 下载软件，并在串口号的下拉列表中选择串口工具所对应的串口号；
选择 MCU 单片机型号；
选择用户程序运行的 IRC 频率，制作仿真芯片时选择的频率与所仿真的用户程序所设置的频率一致，才能达到真实的运行效果。

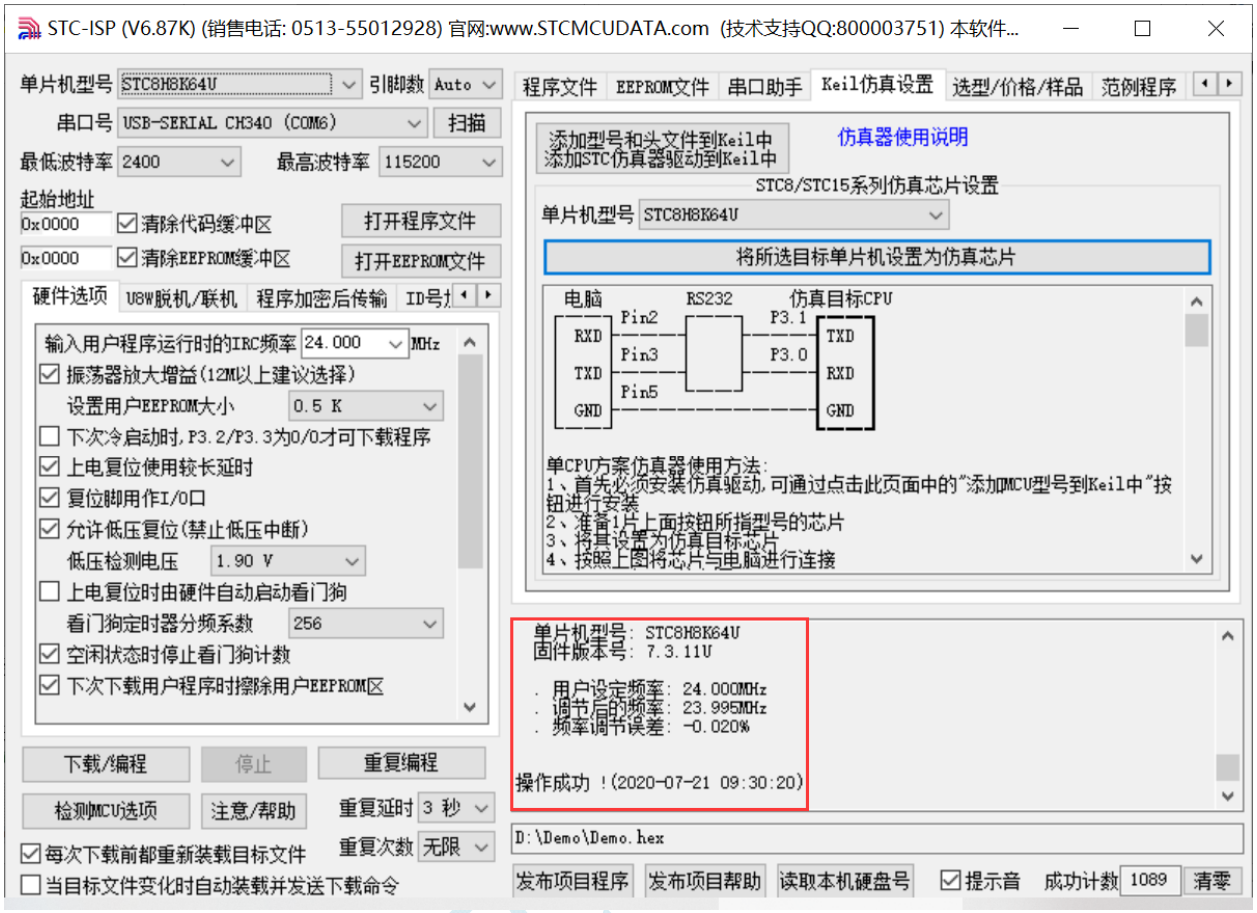


然后在软件右边功能区的“Keil 仿真设置”页面中点击“将所选目标单片机设置为仿真芯片”按钮，按下后会出现如下画面：



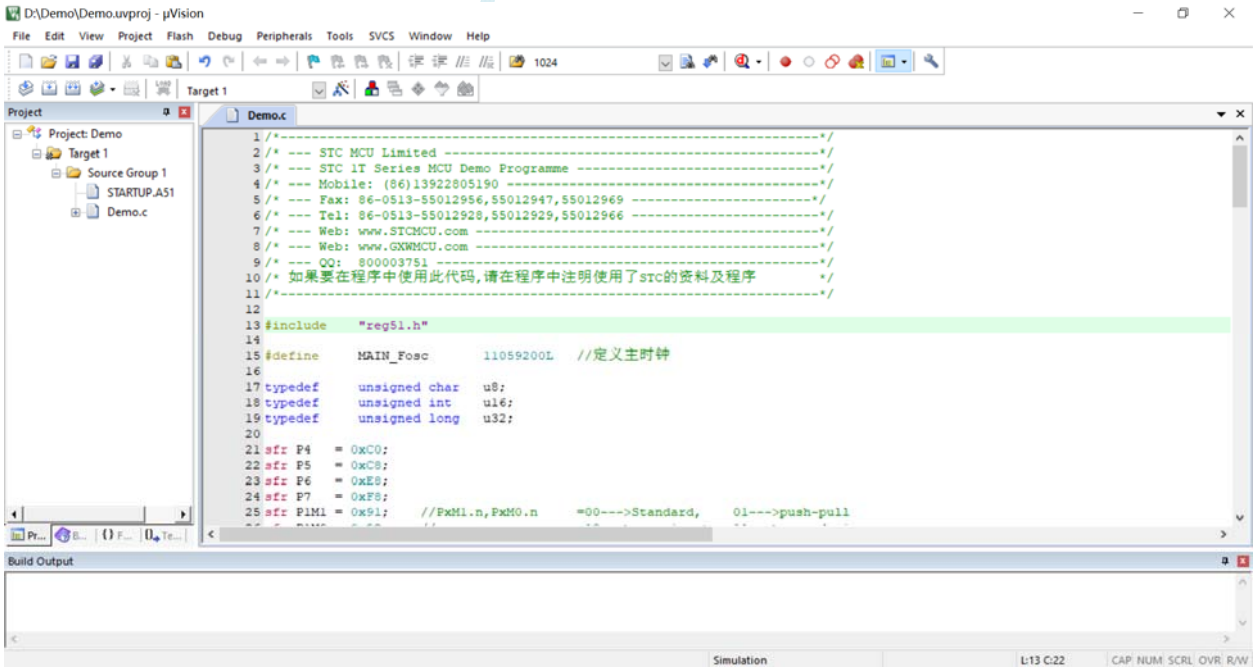
接下来需要按一下 STC 通用 USB 转串口工具上的“**电源开关**”给 MCU 供电【冷启动】，即可开始制作仿真芯片。

若设置成功，会出现如下的画面：



到此，仿真芯片便制作成功了。

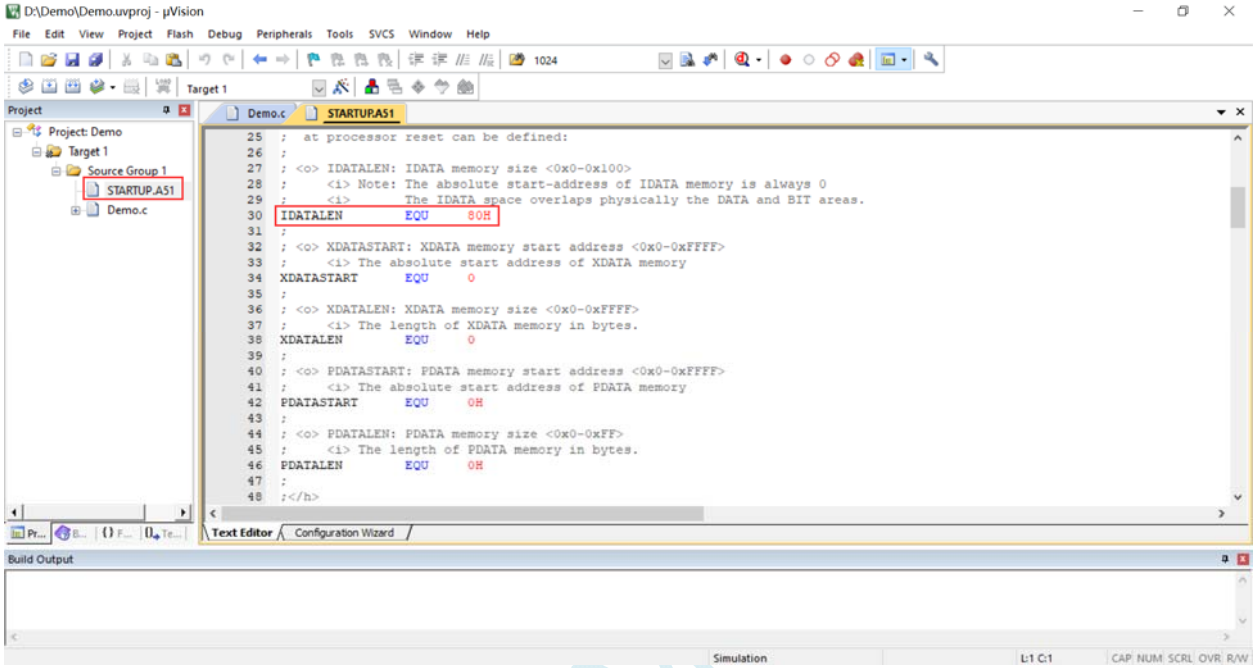
接下来我们打开一个项目进行仿真：



然后进行下面的项目设置：

附加说明一点：

当创建的是 C 语言项目，且有将启动文件“STARTUP.A51”添加到项目中时，里面有一个命名为“IDATALEN”的宏定义，它是用来定义 IDATA 大小的一个宏，默认值是 128，即十六进制的 80H，同时它也是启动文件中需要初始化为 0 的 IDATA 的大小。所以当 IDATA 定义为 80H，那么 STARTUP.A51 里面的代码则会将 IDATA 的 00-7F 的 RAM 初始化为 0；同样若将 IDATA 定义为 0FFH，则会将 IDATA 的 00-FF 的 RAM 初始化为 0。



我们所选的 STC8H 系列的单片机的 IDATA 大小为 256 字节（00-7F 的 DATA 和 80H-FFH 的 IDATA），但由于在 RAM 的最后 17 个字节有写入 ID 号以及相关的测试参数，若用户在程序中需要使用这一部分数据，则一定不要将 IDATALEN 定义为 256。

按下快捷键“Alt+F7”或者选择菜单“Project”中的“Option for Target ‘Target1’”

在“Option for Target ‘Target1’”对话框中对项目进行配置：

第 1 步、进入到项目的设置页面，选择“Debug”设置页；

第 2 步、选择右侧的硬件仿真“Use …”；

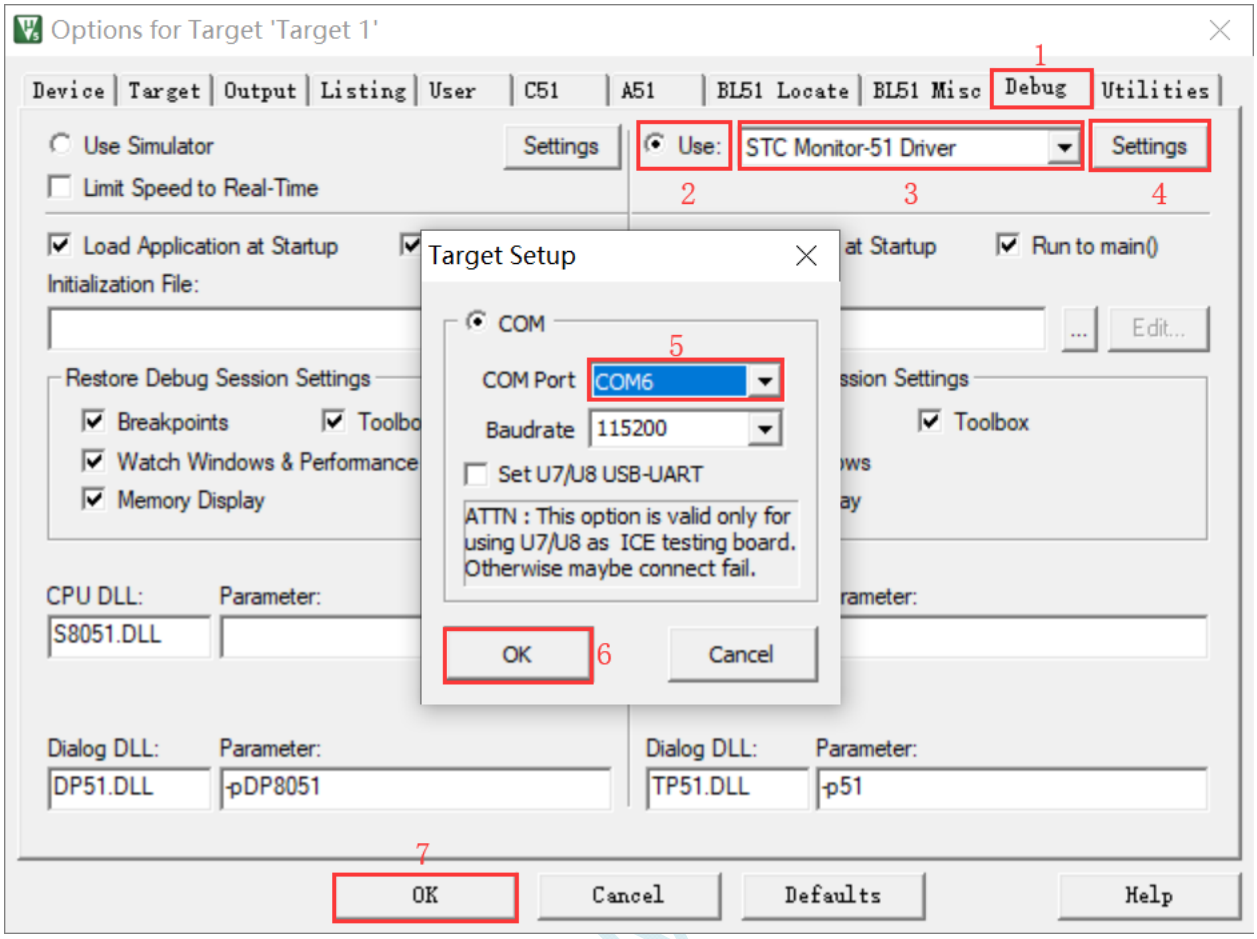
第 3 步、在仿真驱动下拉列表中选择“STC Monitor-51 Driver”项；

第 4 步、点击“Settings”按钮，进入串口的设置画面；

第 5 步、对串口的端口号和波特率进行设置，串口号要选择 STC 通用 USB 转串口工具所对应的串口，波特率一般选择 115200 或者 57600。

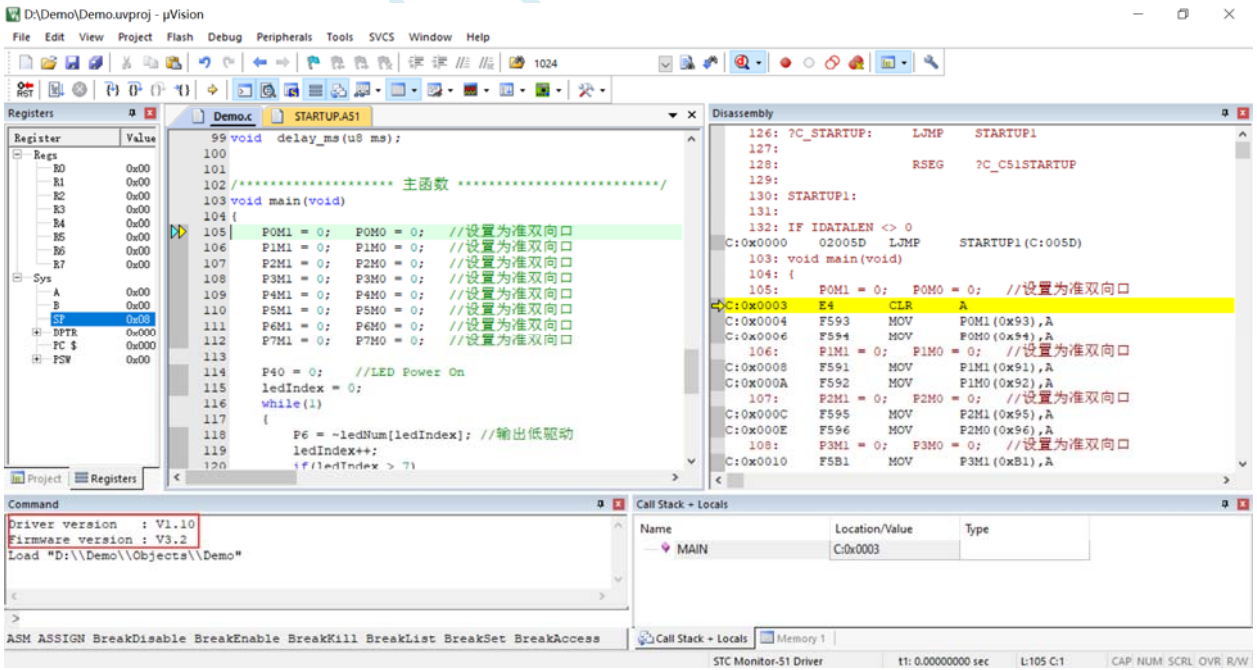
确定完成仿真设置。

详细步骤如下图所示：

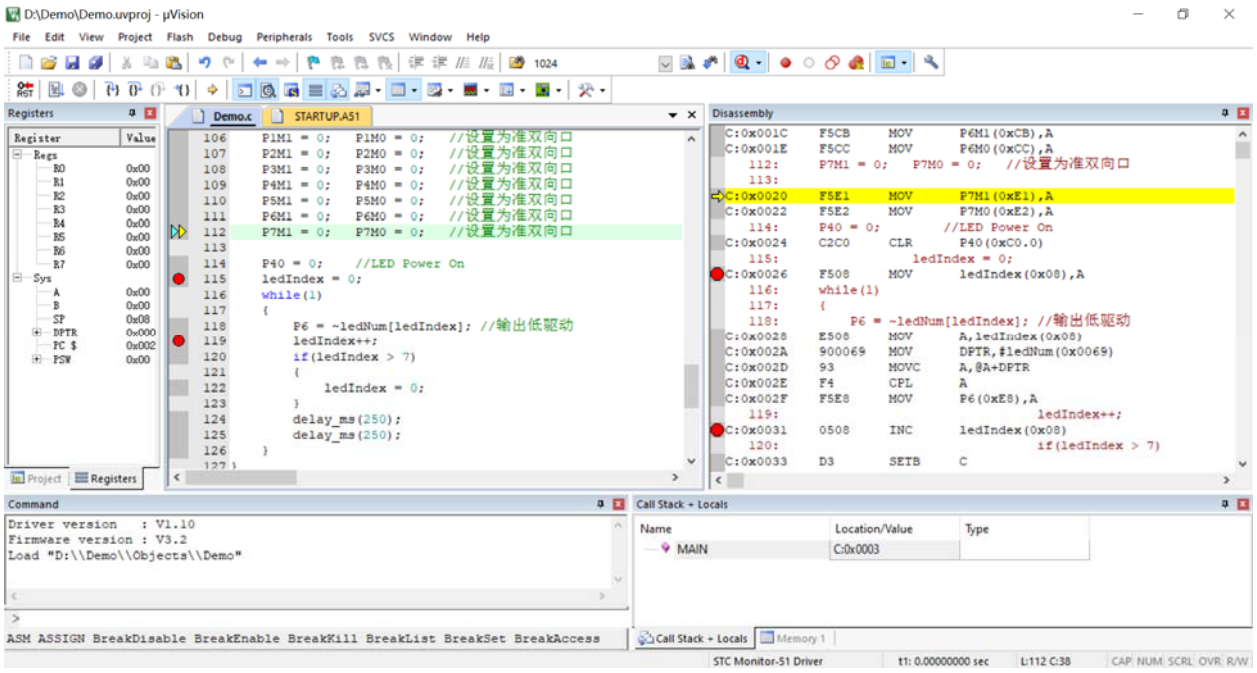


完成了上面所有的工作后，即可在 Keil 软件中按“Ctrl+F5”开始仿真调试。

若硬件连接无误的话，将会进入到类似于下面的调试界面，并在命令输出窗口显示当前的仿真驱动版本号和当前仿真监控代码固件的版本号，如下图所示：



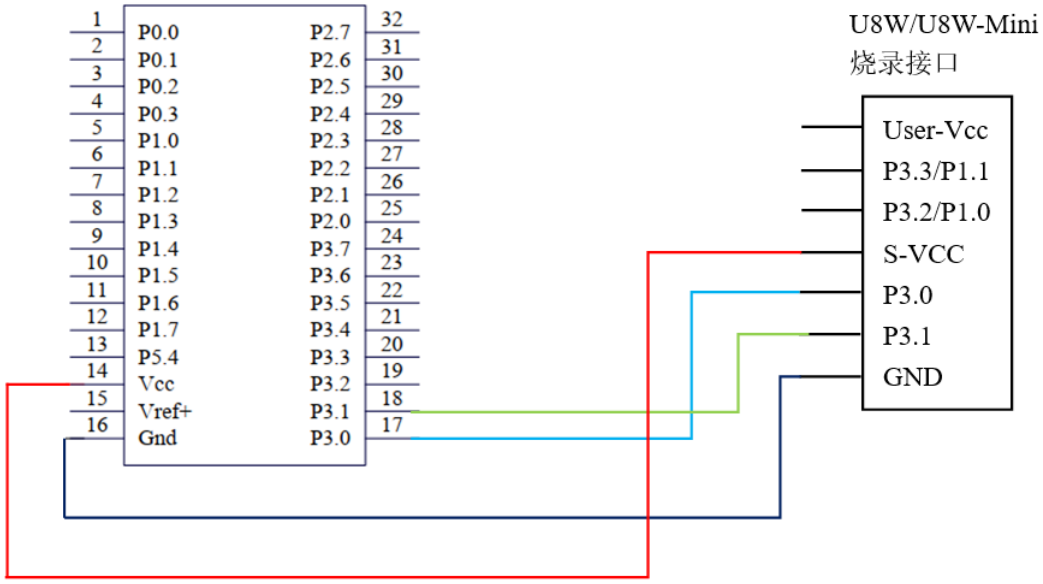
仿真调试过程中，可执行复位、全速运行、单步运行、设置断点等多中操作。



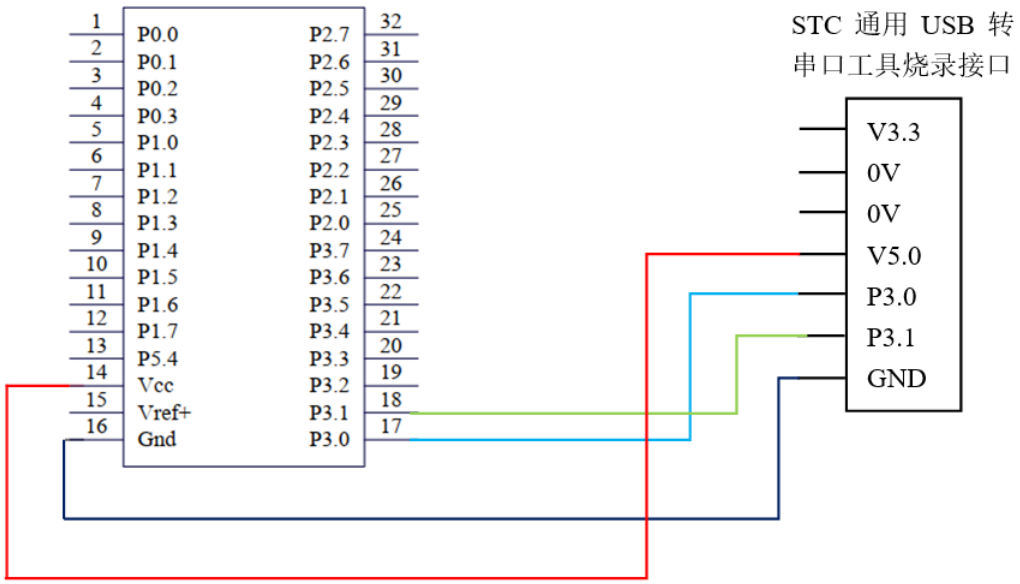
如上图所示，可在程序中设置多个断点，断点设置的个数目前最大允许 20 个（理论上可设置任意个，但是断点设置得过多会影响调试的速度）。

F.5 应用线路图

F.5.1 U8W工具应用参考线路图

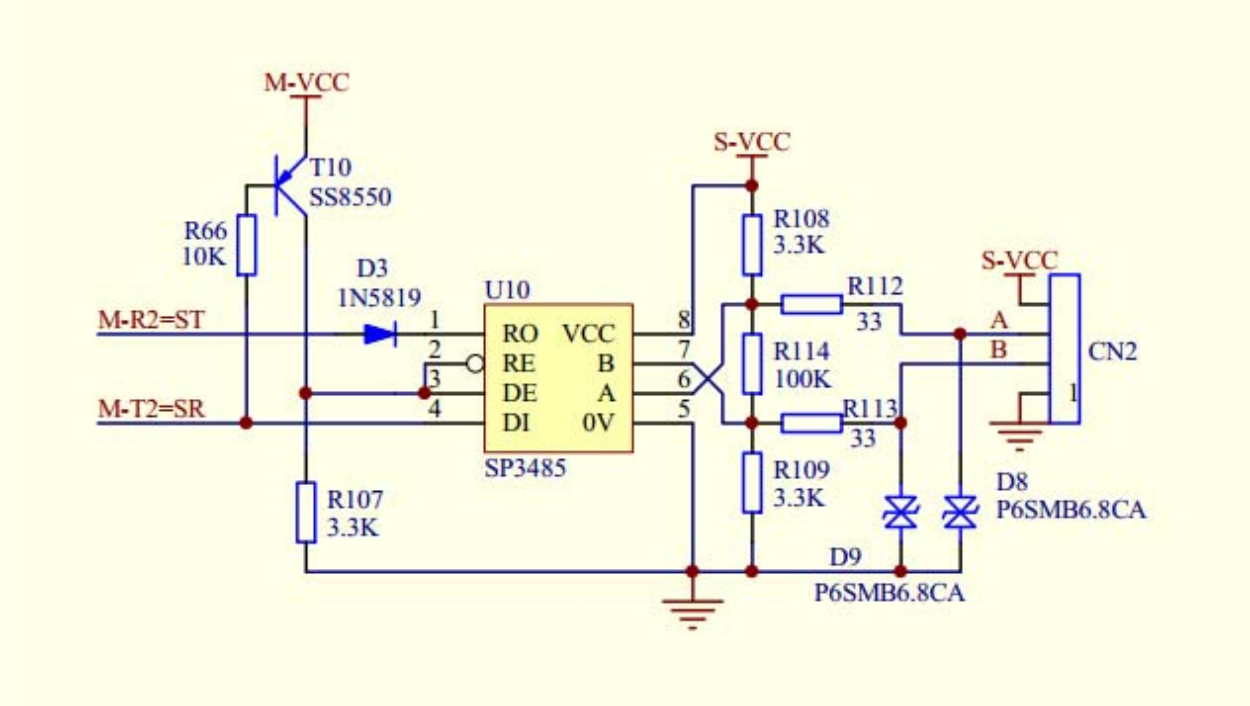


F.5.2 STC通用USB转串口工具应用参考线路图


1 P0.0
2 P0.1
3 P0.2
4 P0.3
5 P1.0
6 P1.1
7 P1.2
8 P1.3
9 P1.4
10 P1.5
11 P1.6
12 P1.7
13 P5.4
14 Vcc
15 Vref+
16 Gnd
32 P2.7
31 P2.6
30 P2.5
29 P2.4
28 P2.3
27 P2.2
26 P2.1
25 P2.0
24 P3.7
23 P3.6
22 P3.5
21 P3.4
20 P3.3
19 P3.2
18 P3.1
17 P3.0
STC 通用 USB 转
串口工具烧录接口
V3.3
0V
0V
V5.0
P3.0
P3.1
GND

STC MCU

附录G U8W下载工具中RS485 部分线路图



BOM 清单：

标号	型号	封装	备注
U10	SP3485EN	SOP8	RS485 芯片
R66	10K	0603	电阻
R107	3.3K	0603	电阻
R108	3.3K	0603	电阻
R109	3.3K	0603	电阻
R112	33R	0603	电阻
R113	33R	0603	电阻
R114	100K	0603	电阻
T10	SS8550	SOT-23	PNP 三极管
D3	1N5819	0603	肖特基二极管
D8	P6SMB6.8CA	DO-214AA	TVS 二极管
D9	P6SMB6.8CA	DO-214AA	TVS 二极管
CN2		SIP4	通信接口

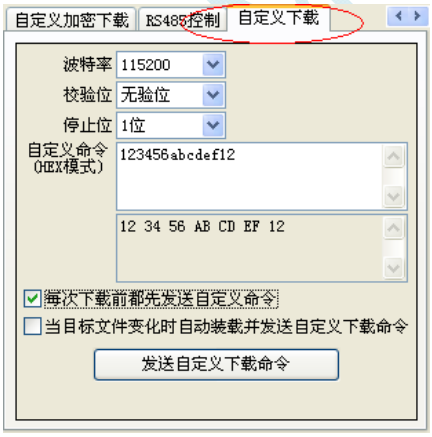
附录H 运行用户程序时收到用户命令后自动启动ISP下载(不停电)

“用户自定义下载”与“用户自定义加密下载”是两种完全不同功能。相对用户自定义加密下载的功能而言，用户自定义下载的功能要简单一些。

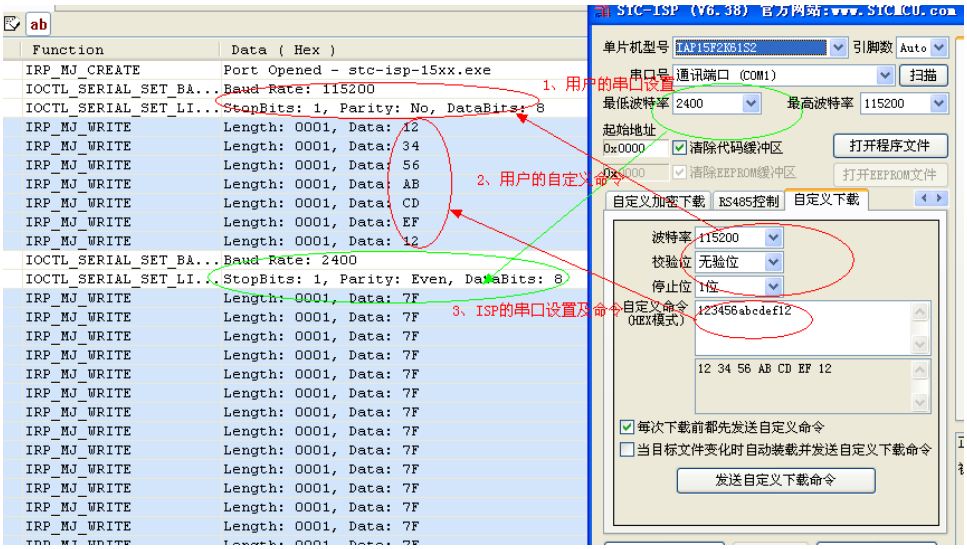
具体的功能为：电脑或脱机下载板在开始发送真正的 ISP 下载编程握手命令前，先发送用户自定义的一串命令（关于这一串串口命令，用户可以根据自己在应用程序中的串口设置来设置波特率、校验位以及停止位），然后再立即发送 ISP 下载编程握手命令。

“用户自定义下载”这一功能主要是在项目的早期开发阶段，实现不断电（不用给目标芯片重新上电）即可下载用户代码。具体的实现方法是：用户需要在自己的程序中加入一段检测自定义命令的代码，当检测到后，执行一句“MOV IAP_CONTR,#60H”的汇编代码或者“IAP_CONTR = 0x60;”的 C 语言代码，MCU 就会自动复位到 ISP 区域执行 ISP 代码。

如下图所示，将自定义命令设置为波特率为 115200、无校验位、一位停止位的命令序列：0x12、0x34、0x56、0xAB、0xCD、0xEF、0x12。当勾选上“每次下载前都先发送自定义命令”的选项后，即可实现自定义下载功能



点击“发送自定义下载命令”或者点击界面左下角的“下载/编程”按钮，应用程序便会发送如下所示的串口数据


Function
Data (Hex)
IRP_MJ_CREATE Port Opened - stc-isp-15xx.exe
IOCTL_SERIAL_SET_BA... Baud Rate: 115200
IOCTL_SERIAL_SET_LI... StopBits: 1, Parity: No, DataBits: 8
IRP_MJ_WRITE Length: 0001, Data: 12
IRP_MJ_WRITE Length: 0001, Data: 34
IRP_MJ_WRITE Length: 0001, Data: 56
IRP_MJ_WRITE Length: 0001, Data: AB
IRP_MJ_WRITE Length: 0001, Data: CD
IRP_MJ_WRITE Length: 0001, Data: EF
IRP_MJ_WRITE Length: 0001, Data: 12
IOCTL_SERIAL_SET_BA... Baud Rate: 2400
IOCTL_SERIAL_SET_LI... StopBits: 1, Parity: Even, DataBits: 8
IRP_MJ_WRITE Length: 0001, Data: 7F
1、用户的串口设置
2、用户的自定义命令
3、ISP的串口设置及命令
单片机型号
引脚数
串口号
通讯端口 (COM1)
扫描
最低波特率
最高波特率
起始地址
清除代码缓冲区
打开程序文件
清除EEPROM缓冲区
打开EEPROM文件
自定义加密下载
RS485控制
自定义下载
波特率
校验位
无验位
停止位
1位
自定义命令 (HEX模式)
123456abcdef12
12 34 56 AB CD EF 12
每次下载前都先发送自定义命令
当目标文件变化时自动装载并发送自定义下载命令
发送自定义下载命令

STC MCU

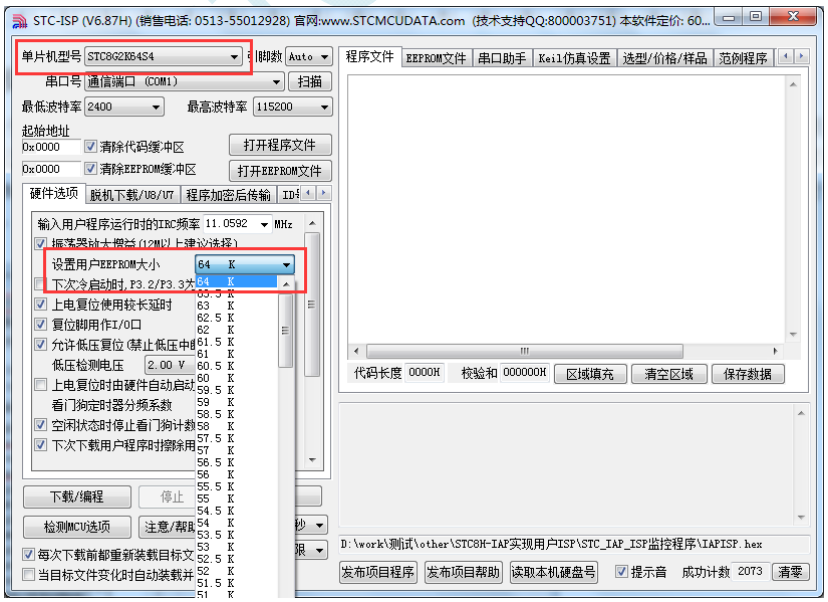
附录I 使用STC的IAP系列单片机开发自己的ISP程序

随着 IAP（In-Application-Programming）技术在单片机领域的不断发展，给应用系统程序代码升级带来了极大的方便。STC 的串口 ISP（In-System-Programming）程序就是使用 IAP 功能来对用户的程序进行在线升级的，但是出于对用户代码的安全着想，底层代码和上层应用程序都没有开源，为此 STC 推出了 IAP 系列单片机，即整颗 MCU 的 Flash 空间，用户均可在自己的程序中进行改写，从而使得有用户需要开发自己的 ISP 程序的想法得以实现。

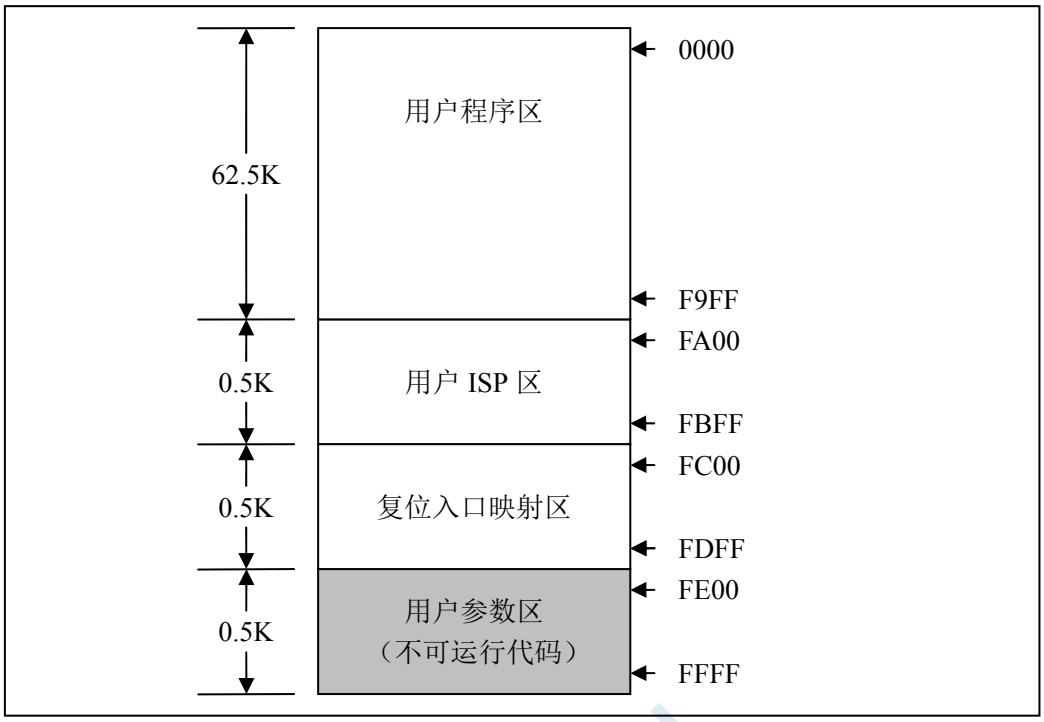
STC8G 系列单片机中的所有可以在 ISP 下载时用户自定义 EEPROM 大小的型号均为 IAP 系列单片机。目前 STC8H 系列有如下型号的单片机为 IAP 系列：STC8G1K12-20/16PIN、STC8G1K17-20/16PIN、STC8G1K12A-8Pin 、 STC8G1K17A-8Pin 、 STC8G1K12-8Pin 、 STC8G1K17-8Pin 、 STC8G1K12T 、 STC8G1K17T、STC8G2K64S2、STC8G2K64S4。本文以 STC8G2K64S4 为例，详细说明使用 STC 的 IAP 单片机开发用户自己的 ISP 程序的方法，并给出了基于 Keil 环境的汇编和 C 源码。

第一步：内部 FLASH 规划

由于 STC8G 系列的 IAP 型号单片机的 EEPROM 是在 ISP 下载时用户自己设置的，所以若用户需要实现自己的 ISP，则在下载用户自己的 ISP 程序时，需要按照下图是方式，将全部的 64K 都设置为 EEPROM，让用户程序空间和 EEPROM 空间完全重合，这样才能实现用户对自己程序空间进行修改和更新。

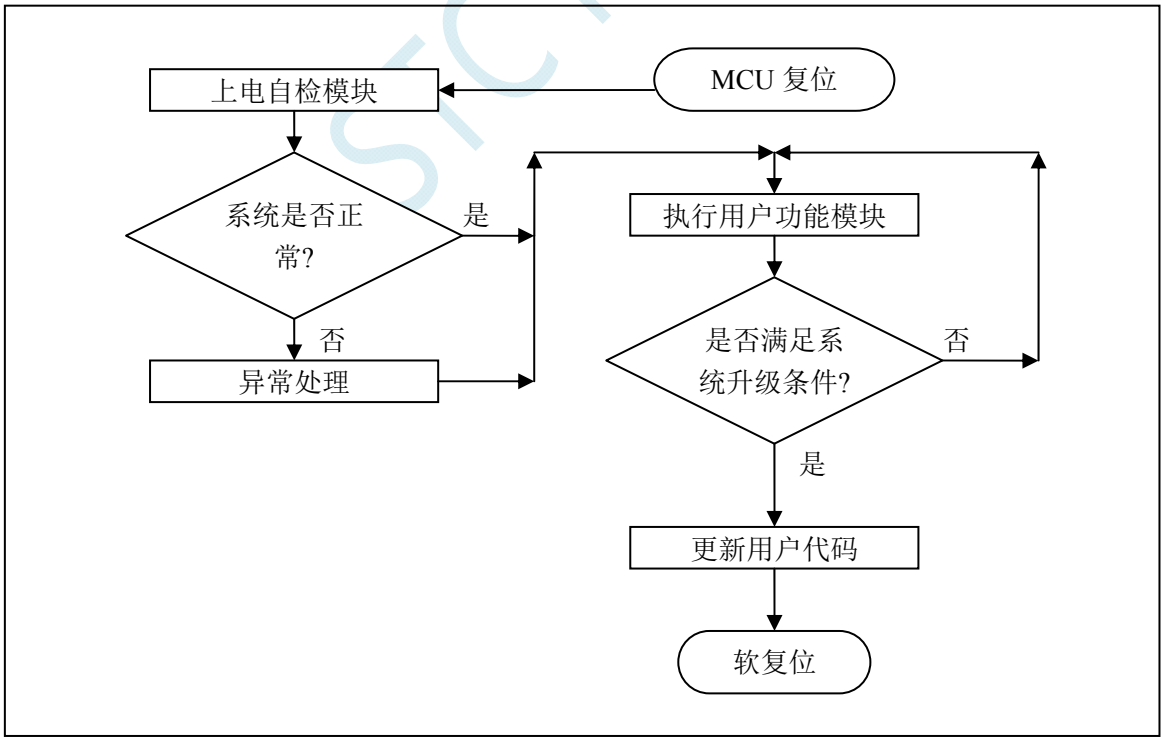


下面假设用户已将整个的 64K 的程序空间已全部设置为 EEPROM，现将整个 64K 程序空间作如下划分：



FLASH 空间中，从地址 0000H 开始的连续 62.5K 字节的空间为用户程序区。当满足特定的下载条件时，需要用户将 PC 跳转到用户 ISP 程序区，此时可对用户程序区进行擦除和改写，以达到更新用户程序的目的。

第二步、程序的基本框架



第三步、下位机固件程序说明

下位机固件程序包括两部分：ISP（ISP 代码）和 AP（用户代码）

ISP 代码（汇编代码）

```
;测试工作频率为 11.0592MHz

UARTBAUD    EQU     0FFE8H          ;定义串口波特率      (65536-11059200/4/115200)

AUXR        DATA    08EH            ;附加功能控制寄存器
WDT_CONTR   DATA    0C1H            ;看门狗控制寄存器
IAP_DATA    DATA    0C2H            ;IAP 数据寄存器
IAP_ADDRH   DATA    0C3H            ;IAP 高地址寄存器
IAP_ADDRL   DATA    0C4H            ;IAP 低地址寄存器
IAP_CMD     DATA    0C5H            ;IAP 命令寄存器
IAP_TRIG    DATA    0C6H            ;IAP 命令触发寄存器
IAP_CONTR   DATA    0C7H            ;IAP 控制寄存器
IAP_TPS     DATA    0F5H            ;IAP 等待时间控制寄存器

ISPCODE     EQU     0FA00H          ;ISP 模块入口地址(1 页),同时也是外部接口地址
APENTRY     EQU     0FC00H          ;应用程序入口地址数据(1 页)

            ORG     0000H

            LJMP    ISP_ENTRY       ;系统复位入口

RESET:
            MOV     SCON,#50H       ;设置串口模式(8 位数据位,无校验位)
            MOV     AUXR,#40H       ;定时器 1 为 1T 模式
            MOV     TMOD,#00H       ;定时器 1 工作于模式 0(16 位重装载)
            MOV     TH1,#HIGH UARTBAUD  ;设置重载值
            MOV     TL1,#LOW UARTBAUD
            SETB    TR1             ;启动定时器 1
NEXT1:
            MOV     R0,#16
NEXT2:
            JNB     RI,$            ;等待串口数据
            CLR     RI
            MOV     A,SBUF
            CJNE    A,#7FH,NEXT1    ;判断是否为 7F
            DJNZ    R0,NEXT2
            LJMP    ISP_DOWNLOAD    ;跳转到下载界面

            ORG     ISPCODE

ISP_DOWNLOAD:
            CLR     A
            MOV     PSW,A           ;ISP 模块使用第 0 组寄存器
            MOV     IE,A            ;关闭所有中断
            CLR     RI              ;清除串口接收标志
            SETB    TI              ;置串口发送标志
            CLR     TR0
            MOV     SP,#5FH         ;设置堆栈指针

            MOV     A,#5AH          ;返回 5A  55 到 PC,表示 ISP 擦除模块已准备就绪
            LCALL   ISP_SENDUART
            MOV     A,#055H
            LCALL   ISP_SENDUART
            LCALL   ISP_RECVACK     ;接收应答数据

            MOV     IAP_ADDRL,#0    ;首先在第 2 页起始地址写 "LJMP ISP_ENTRY"指令
            MOV     IAP_ADDRH,#02H
            LCALL   ISP_ERASEIAP
```

```
                MOV     A,#02H
                LCALL   ISP_PROGRAMIAP          ;编程用户代码复位向量代码
                MOV     A,#HIGH                 ISP_ENTRY
                LCALL   ISP_PROGRAMIAP          ;编程用户代码复位向量代码
                MOV     A,#LOW ISP_ENTRY
                LCALL   ISP_PROGRAMIAP          ;编程用户代码复位向量代码

                MOV     IAP_ADDRL,#0            ;用户代码地址从 0 开始
                MOV     IAP_ADDRH,#0
                LCALL   ISP_ERASEIAP
                MOV     A,#02H
                LCALL   ISP_PROGRAMIAP          ;编程用户代码复位向量代码
                MOV     A,#HIGH                 ISP_ENTRY
                LCALL   ISP_PROGRAMIAP          ;编程用户代码复位向量代码
                MOV     A,#LOW ISP_ENTRY
                LCALL   ISP_PROGRAMIAP          ;编程用户代码复位向量代码

                MOV     IAP_ADDRL,#0            ;新代码缓冲区地址
                MOV     IAP_ADDRH,#02H
                MOV     R7,#124                 ;擦除 62.5K 字节
ISP_ERASEAP:
                LCALL   ISP_ERASEIAP
                INC     IAP_ADDRH               ;目标地址+512
                INC     IAP_ADDRH
                DJNZ    R7,ISP_ERASEAP          ;判断是否擦除完成

                MOV     IAP_ADDRL,#LOW APENTRY
                MOV     IAP_ADDRH,#HIGH APENTRY
                LCALL   ISP_ERASEIAP

                MOV     A,#5AH                  ;返回 5A  A5 到 PC,表示 ISP 编程模块已准备就绪
                LCALL   ISP_SENDUART
                MOV     A,#0A5H
                LCALL   ISP_SENDUART
                LCALL   ISP_RECVACK             ;接收应答数据

                LCALL   ISP_RECVUART            ;接收长度高字节
                MOV     R0,A
                LCALL   ISP_RECVUART            ;接收长度低字节
                MOV     R1,A
                CLR     C                       ;将总长度-3
                MOV     A,#03H
                SUBB    A,R1
                MOV     DPL,A
                CLR     A
                SUBB    A,R0
                MOV     DPH,A                   ;总长度补码存入 DPTR

                LCALL   ISP_RECVUART            ;映射用户代码复位入口代码到映射区
                LCALL   ISP_PROGRAMIAP          ;0000
                LCALL   ISP_RECVUART
                LCALL   ISP_PROGRAMIAP          ;0001
                LCALL   ISP_RECVUART
                LCALL   ISP_PROGRAMIAP          ;0002

                MOV     IAP_ADDRL,#03H          ;用户代码起始地址
                MOV     IAP_ADDRH,#00H
ISP_PROGRAMNEXT:
                LCALL   ISP_RECVUART            ;接收代码数据
```

```
                LCALL   ISP_PROGRAMIAP          ;编程到用户代码区
                INC     DPTR
                MOV     A,DPL
                ORL     A,DPH
                JNZ     ISP_PROGRAMNEXT         ;长度检测

ISP_SOFTRESET:
                MOV     IAP_CONTR,#20H          ;软件复位系统
                SJMP    $


ISP_ENTRY:
                MOV     WDT_CONTR,#17H          ;清看门狗
                MOV     IAP_CONTR,#80H          ;使能 IAP 功能
                MOV     IAP_TPS,#11             ;设置 IAP 等待时间参数
                MOV     IAP_ADDRL,#LOW ISP_DOWNLOAD
                MOV     IAP_ADDRH,#HIGH ISP_DOWNLOAD
                MOV     IAP_DATA,#00H           ;测试数据 1
                MOV     IAP_CMD,#1              ;读命令
                MOV     IAP_TRIG,#5AH           ;触发 ISP 命令
                MOV     IAP_TRIG,#0A5H
                MOV     A,IAP_DATA
                CJNE    A,#0E4H,ISP_ENTRY       ;若无法读出数据则需要等待电压稳定
                INC     IAP_ADDRL               ;测试地址 FC01H
                MOV     IAP_DATA,#45H           ;测试数据 2
                MOV     IAP_CMD,#1              ;读命令
                MOV     IAP_TRIG,#5AH           ;触发 ISP 命令
                MOV     IAP_TRIG,#0A5H
                MOV     A,IAP_DATA
                CJNE    A,#0F5H,ISP_ENTRY       ;若无法读出数据则需要等待电压稳定

                MOV     SCON,#50H               ;设置串口模式(8 位数据位,无校验位)
                MOV     AUXR,#40H               ;定时器 1 为 1T 模式
                MOV     TMOD,#00H               ;定时器 1 工作于模式 0(16 位重装载)
                MOV     TH1,#HIGH UARTBAUD      ;设置重载值
                MOV     TL1,#LOW UARTBAUD
                SETB    TR1                     ;启动定时器 1
                SETB    TR0

                LCALL   ISP_RECVUART            ;检测是否有串口数据
                JC      GOTOAP
                MOV     R0,#16
ISP_CHECKNEXT:
                LCALL   ISP_RECVUART            ;接收同步数据
                JC      GOTOAP
                CJNE    A,#7FH,GOTOAP           ;判断是否为 7F
                DJNZ    R0,ISP_CHECKNEXT
                MOV     A,#5AH                  ;返回 5A  69 到 PC,表示 ISP 模块已准备就绪
                LCALL   ISP_SENDUART
                MOV     A,#69H
                LCALL   ISP_SENDUART
                LCALL   ISP_RECVACK             ;接收应答数据
                LJMP    ISP_DOWNLOAD            ;跳转到下载界面
GOTOAP:
                CLR     A                       ;将 SFR 恢复为复位值
                MOV     TCON,A
                MOV     TMOD,A
                MOV     TL0,A
                MOV     TH0,A
```

```
                MOV     TL1,A
                MOV     TH1,A
                MOV     SCON,A
                MOV     AUXR,A
                LJMP    APENTRY             ;正常运行用户程序

ISP_RECVACK:
                LCALL   ISP_RECVUART
                JC      GOTOAP
                XRL     A,#7FH
                JZ      ISP_RECVACK         ;跳过同步数据
                CJNE    A,#25H,GOTOAP       ;应答数据 1 检测
                LCALL   ISP_RECVUART
                JC      GOTOAP
                CJNE    A,#69H,GOTOAP       ;应答数据 2 检测
                RET

ISP_RECVUART:
                CLR     A
                MOV     TL0,A               ;初始化超时定时器
                MOV     TH0,A
                CLR     TF0
                MOV     WDT_CONTR,#17H      ;清看门狗
ISP_RECVWAIT:
                JBC     TF0,ISP_RECVTIMEOUT ;超时检测
                JNB     RI,ISP_RECVWAIT     ;等待接收完成
                MOV     A,SBUF              ;读取串口数据
                CLR     RI                  ;清除标志
                CLR     C                   ;正确接收串口数据
                RET
ISP_RECVTIMEOUT:
                SETB    C                   ;超时退出
                RET

ISP_SENDUART:
                MOV     WDT_CONTR,#17H      ;清看门狗
                JNB     TI,ISP_SENDUART     ;等待前一个数据发送完成
                CLR     TI                  ;清除标志
                MOV     SBUF,A              ;发送当前数据
                RET

ISP_ERASEIAP:
                MOV     WDT_CONTR,#17H      ;清看门狗
                MOV     IAP_CMD,#3          ;擦除命令
                MOV     IAP_TRIG,#5AH       ;触发 ISP 命令
                MOV     IAP_TRIG,#0A5H
                NOP
                NOP
                NOP
                NOP
                RET

ISP_PROGRAMIAP:
                MOV     WDT_CONTR,#17H      ;清看门狗
                MOV     IAP_CMD,#2          ;编程命令
                MOV     IAP_DATA,A          ;将当前数据送 IAP 数据寄存器
                MOV     IAP_TRIG,#5AH       ;触发 ISP 命令
                MOV     IAP_TRIG,#0A5H
                NOP
```

```
        NOP
        NOP
        NOP
        MOV     A,IAP_ADDRL             ;IAP 地址+1
        ADD     A,#01H
        MOV     IAP_ADDRL,A
        MOV     A,IAP_ADDRH
        ADDC    A,#00H
        MOV     IAP_ADDRH,A
        RET


        ORG     APENTRY
        LJMP    RESET


        END
```

ISP 代码包括如下外部接口模块：

ISP_DOWNLOAD：程序下载入口地址，绝对地址 FA00H

ISP_ENTRY：上电系统自检程序（系统自动调用）

对于用户程序而言，用户只需要在满足下载条件时，将 PC 值跳转到 ISPPROGRAM （即 FA00H 的绝对地址），即可实现代码更新。

用户代码（C 语言代码）

```
//测试工作频率为 11.0592MHz

#include "reg51.h"

#define   FOSC              11059200L           //系统时钟频率
#define   BAUD              (65536 - FOSC/4/115200)     //定义串口波特率
#define   ISPPROGRAM        0xfa00              //ISP 下载程序入口地址

sfr       AUXR      =       0x8e;               //波特率发生器控制寄存器
sfr       P1M0      =       0x92;
sfr       P1M1      =       0x91;

void (*IspProgram)() = ISPPROGRAM;              //定义指针函数
char cnt7f;                                     //Isp_Check 内部使用的变量

void uart() interrupt 4                         //串口中断服务程序
{
    if (TI) TI = 0;                             //发送完成中断
    if (RI)                                     //接收完成中断
    {
        if (SBUF == 0x7f)
        {
            cnt7f++;
            if (cnt7f >= 16)
            {
                IspProgram();                   //调用下载模块(****重要语句****)
            }
        }
        else
```

```
        {
            cnt7f = 0;
        }
        RI = 0;                                 //清接收完成标志
    }
}

void main()
{
    SCON = 0x50;                                //定义串口模式为8bit 可变,无校验位
    AUXR = 0x40;
    TH1  = BAUD >> 8;
    TL1  = BAUD;
    TR1  = 1;
    ES = 1;                                     //使能串口中断
    EA = 1;                                     //打开全局中断开关

    P1M0 = 0;
    P1M1 = 0;

    while (1)
    {
        P1++;
    }
}
```

用户代码（汇编代码）

```
;测试工作频率为11.0592MHz

UARTBAUD    EQU     0FFE8H          ;定义串口波特率      (65536-11059200/4/115200)
ISPPROGRAM  EQU     0FA00H          ;ISP 下载程序入口地址

AUXR        DATA    08EH            ;附件功能控制寄存器

CNT7F       DATA    60H             ;接收7F 的计数器

            ORG     0000H
            LJMP    START           ;系统复位入口

            ORG     0023H
            LJMP    UART_ISR        ;串口中断入口

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            JNB     TI,CHECKRI      ;检测发送中断
            CLR     TI              ;清除标志
CHECKRI:
            JNB     RI,UARTISR_EXIT ;检测接收中断
            CLR     RI              ;清除标志
            MOV     A,SBUF
            CJNE    A,#7FH,ISNOT7F
            INC     CNT7F
            MOV     A,CNT7F
            CJNE    A,#16,UARTISR_EXIT
            LJMP    ISPPROGRAM      ;调用下载模块(****重要语句****)
ISNOT7F:
            MOV     CNT7F,#0
```

```
UARTISR_EXIT:
        POP     PSW
        POP     ACC
        RETI

START:
        MOV     R0,#7FH             ;清 RAM
        CLR     A
        MOV     @R0,A
        DJNZ    R0,$-1
        MOV     SP,#7FH             ;初始化 SP

        MOV     SCON,#50H           ;设置串口模式(8 位可变,无校验位)
        MOV     AUXR,#15H           ;BRT 工作于 1T 模式,启动 BRT
        MOV     TMOD,#00H           ;定时器 1 工作于模式 0(16 位重装载)
        MOV     TH1,#HIGH UARTBAUD  ;设置重载值
        MOV     TL1,#LOW UARTBAUD
        SETB    TR1                 ;启动定时器 1
        SETB    ES                  ;使能串口中断
        SETB    EA                  ;开中断总开关

MAIN:
        INC     P0
        SJMP    MAIN

        END
```

STC MCU

用户代码可以使用 C 或者汇编语言编写，但对于汇编代码需要注意一点：位于 0000H 的复位入口地址处的指令必须是一个长跳转语句（类似 LJMP START）。在用户代码中，需要设置好串口，并在满足下载条件时，将 PC 值跳转到 ISPPROGRAM （即 FA00H 的绝对地址），以实现代码更新。对于汇编代码，我们可以使用“LJMP 0FA00H”指令进行调用，如下图

```
UARTBAUD    EQU     0FFE8H          ;定义串口波特率 (65536-11059200/4/115200)
ISPPROGRAM  EQU     0FA00H          ;ISP下载程序入口地址

AUXR        DATA    08EH            ;附件功能控制寄存器
```

```
18          CLR     TI                      ;清除标志
19  CHECKRI:
20          JNB     RI,UARTISR_EXIT         ;检测接收中断
21          CLR     RI                      ;清除标志
22          MOV     A,SBUF
23          CJNE    A,#7FH,ISNOT7F
24          INC     CNT7F
25          MOV     A,CNT7F
26          CJNE    A,#16,UARTISR_EXIT
27          LJMP    ISPPROGRAM              ;调用下载模块(****重要语句****)
28  ISNOT7F:
29          MOV     CNT7F,#0
30  UARTISR_EXIT:
31          POP     PSW
32          POP     ACC
33          RETI
34
35  START:
```

在 C 代码中，必须定义一个函数指针变量，并将此变量赋值为 0xFA00，然后再调用，如下图

```
#include "reg51.h"

#define FOSC            11059200L               //系统时钟频率
#define BAUD            (65536 - FOSC/4/115200) //定义串口波特率
#define ISPPROGRAM      0xfa00                  //ISP下载程序入口地址

sfr AUXR    =   0x8e;                           //波特率发生器控制寄存器
sfr P1M0    =   0x92;
sfr P1M1    =   0x91;

void (*IspProgram)() = ISPPROGRAM;              //定义指针函数
char cnt7f;                                     //Isp_Check内部使用的变量

void uart() interrupt 4                         //串口中断服务程序
{
    if (TI) TI = 0;                             //发送完成中断
    if (RI)                                     //接收完成中断
    {
        if (SBUF == 0x7f)
        {
            cnt7f++;
            if (cnt7f >= 16)
            {
                IspProgram();                   //调用下载模块(****重要语句****)
            }
        }
        else
        {
            cnt7f = 0;
        }
        RI = 0;                         //清接收完成标志
    }
```

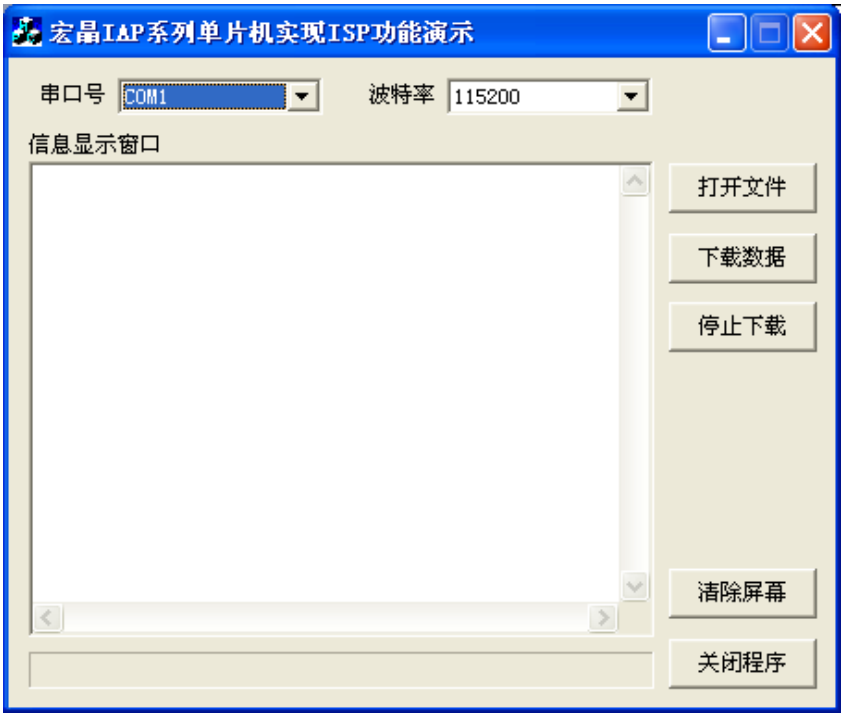
第四步、上位机应用程序说明

上位机的程序是基于 MFC 的对话框项目，对于串口的访问是直接调用 Windows 的 API 函数，而没有使用串口控件，从而省去的控件的注册以及系统版本不兼容的诸多问题。界面较简单，只是为这一功能的实现提供了一个框架，其他的功能及要求均还可以往上面添加。

上位机程序的核心模块是基于类 CISPDlg 的一个友元函数“UINT Download(LPVOID pParam);”，此函数负责与下位机通讯，发送各种通讯命令来完成对用户程序的更新。用户可以根据各自不同的需求增加命令。

第五步、上位机应用程序的使用方法

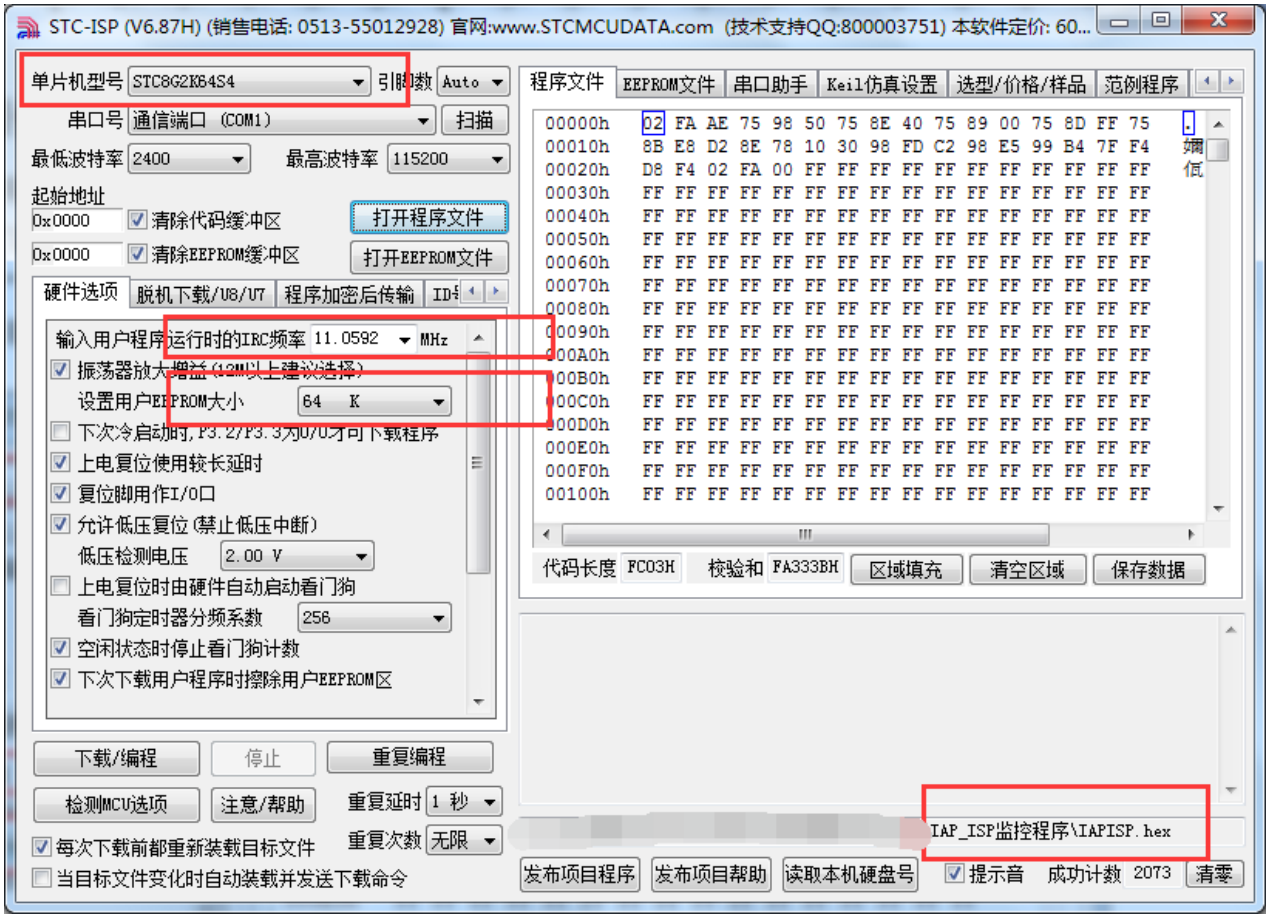
- 打开上位机界面，如下图



- 选择串口号，设置与下位机相同的串口波特率
- 打开要下载的源数据文件，Bin 或者 Intel hex 格式均可以
- 点击“下载数据”按钮即可开始下载数据

第六步、下位机固件程序的使用方法

下位机的目标文件有两个“IAPISP.hex”和“AP.hex”，对于一块新的单片机，第一次必须使用宏晶科技的 ISP 下载工具将“IAPISP.hex”写入到芯片内，如下图所示。之后再更新便不再需要写“IAPISP.hex”这个文件了，附件中的“AP.hex”只是一个用户程序的模板，当满足下载条件时，用户只需要将 PC 值跳转到 FA00H 的地址，即可实现代码更新。



附录J 用户程序复位到系统区进行ISP下载的方法（不停电）

当项目处于开发阶段时，需要反复的下载用户代码到目标芯片中进行代码验证，而 STC 的单片机进行正常的 ISP 下载都需要对目标芯片进行重新上电，从而会使得项目在开发阶段比较繁琐。为此 STC 单片机增加了一个特殊功能寄存器 IAP_CONTR，当用户向此寄存器写入 0x60，即可实现软件复位到系统区，进而实现不停电就可进行 ISP 下载。

但是用户如何判断是否正在进行 ISP 下载？何时向寄存器 IAP_CONTR 写 0x60 触发软复位？就这两个问题，下面分别介绍四种判断方法：

使用 P3.0 口检测串口起始信号

STC 单片机的串口 ISP 固定使用 P3.0 和 P3.1 两个端口，当 ISP 下载软件开始下载时，会发送握手命令到单片机的 P3.0 口。若用户的 P3.0 和 P3.1 只是专门用于 ISP 下载，则可使用 P3.0 口检测串口的起始信号来判断 ISP 下载。

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         IAP_CONTR   =   0xc7;
sfr         P3M0        =   0xb2;
sfr         P3M1        =   0xb1;

sbit        P30         =   P3^0;

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;
    P30 = 1;

    while (1)
    {
        if (!P30) IAP_CONTR = 0x60;          //P3.0 的低电平即为串口起始信号
                                             //软件复位到系统区

        ...                                  //用户代码
    }
}
```

使用 P3.0/INT4 口的下降沿中断，检测串口起始信号

方法 B 与方法 A 类似，不同在于方法 A 使用的是查询方式，方法 B 使用中断方式。因为 STC 单片机的 P3.0 口为 INT4 的中断口。

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         IAP_CONTR   =   0xc7;
sfr         INTCLKO     =   0x8f;
sfr         P3M0        =   0xb2;
sfr         P3M1        =   0xb1;

void Int4Isr() interrupt 16                 //INT4 中断服务程序
{
    IAP_CONTR = 0x60;                       //串口起始信号触发 INT4 中断
                                            //软件复位到系统区
}

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    INTCLKO |= 0x40;                        //使能 INT4 中断
    EA = 1;

    while (1)
    {
        ...                                 //用户代码
    }
}
```

使用 P3.0/RxD 口的串口接收，检测 ISP 下载软件发送的 7F

方法 A 与方法 B 都非常简单，但容易受干扰，如果 P3.0 口有任何一个干扰信号，都会触发软件复位，所以方法 C 是对串口数据进行校验。

STC 的 ISP 下载软件进行 ISP 下载时，首先都会使用最低波特率（一般是 2400）+偶校验 9+1 位停止位连续发送握手命令 7F，因此用户可以在程序中，将串口设置为 9 位数据位+2400 波特率，然后持续检测 7F，比如连续检测到 8 个 7F 表示可确定需要进行 ISP 下载，此时再触发软件复位。

C 语言代码

```
//测试工作频率为11.0592MHz
```

```
#include "reg51.h"
#include "intrins.h"

#define   FOSC          11059200UL
#define   BR2400        (65536 - FOSC / 4 / 2400)

sfr       IAP_CONTR   =   0xc7;
sfr       AUXR        =   0x8e;
sfr       P3M0        =   0xb2;
sfr       P3M1        =   0xb1;

char cnt7f;

void UartIsr() interrupt 4                          //串口中断服务程序
{
    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;
        if ((SBUF == 0x7f) && (RB8 == 1))           //ISP 下载软件发送的握手命令 7F
                                                    //7F 的偶校验位为 1
        {
            if (++cnt7f == 8)                       //当连续检测到 8 个 7F 后
                IAP_CONTR = 0x60;                   //复位到系统区
        }
        else
        {
            cnt7f = 0;
        }
    }
}

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    SCON = 0xd0;                                    //设置串口为 9 位数据位
    TMOD = 0x00;
    AUXR = 0x40;
    TH1 = BR2400 >> 8;                              //设置串口波特率为 2400
    TL1 = BR2400;
    TR1 = 1;
    ES = 1;
    EA = 1;

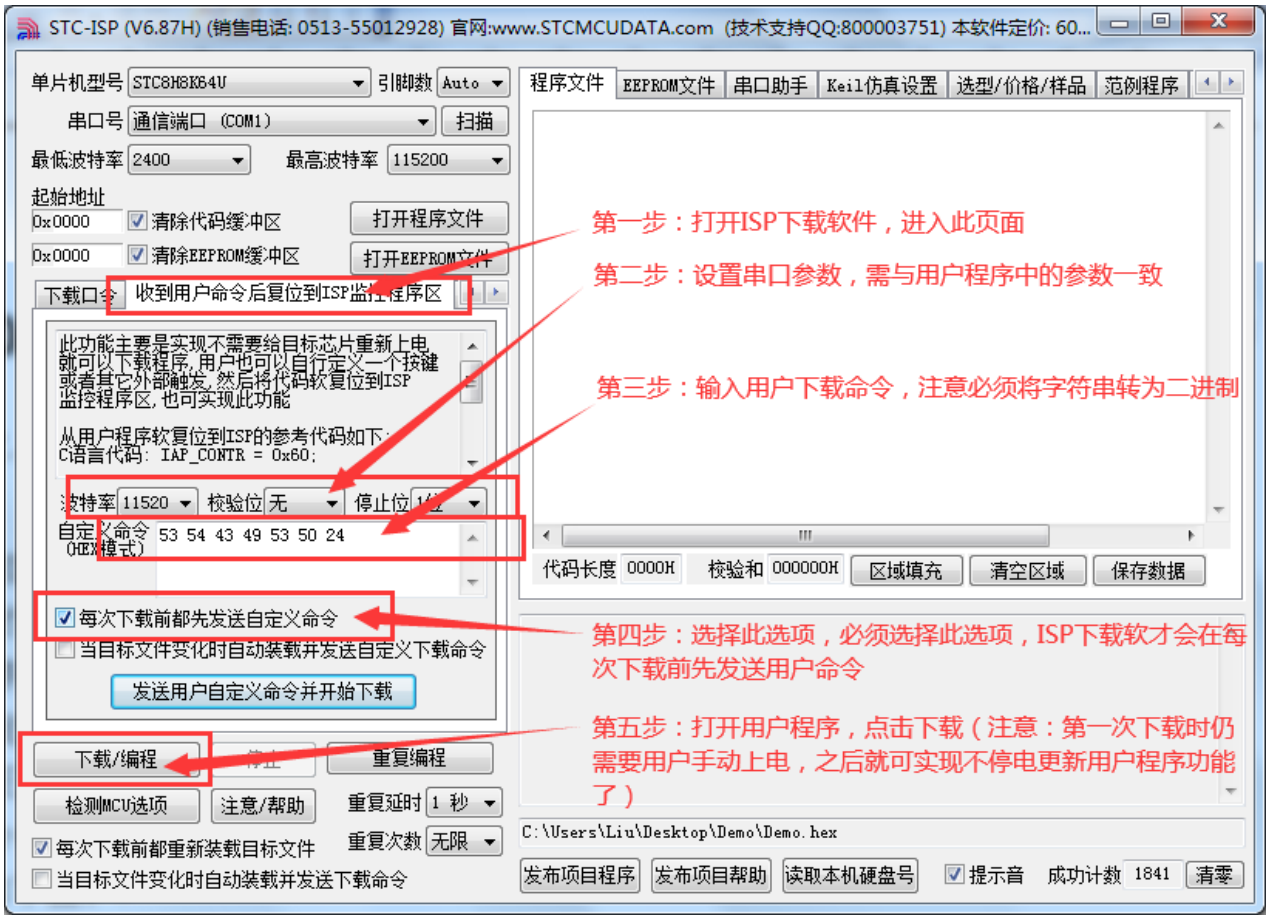
    cnt7f = 0;

    while (1)
    {
        ...                                         //用户代码
    }
}
```

使用 P3.0/RxD 串口接收，检测 ISP 下载软件发送的用户下载命令

如果用户代码中需要使用串口进行通信，则上面的 3 中方法可能都不太适用，此时可以使用 STC 的 ISP 下载软件提供的接口，定制一组专用的用户下载命令（可指定波特率、校验位和停止位），若使能此功能，ISP 下载软件在进行 ISP 下载前，会使用用户指定的波特率、校验位和停止位发送用户下载命令，然后再发送握手命令。用户只需要在自己的代码中监控串口命令序列，当检测到有正确的用户下载命令时，软件复位到系统区即可实现不停电进行 ISP 功能。

下面假设用户下载命令为字符串“STCISP$”，串口设置为波特率 115200，无校验位和 1 位停止位。ISP 下载软件中的设置如下图：



用户示例代码如下：

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC          11059200UL
#define   BR115200      (65536 - FOSC / 4 / 115200)

sfr       IAP_CONTR   =   0xc7;
sfr       AUXR        =   0x8e;
sfr       P3M0        =   0xb2;
```

```
sfr         P3M1            =       0xb1;

char stage;

void UartIsr() interrupt 4                              //串口中断服务程序
{
    char dat;

    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;

        dat = SBUF;
        switch (stage)
        {
        case 0:
        default:
L_Check1st:
            if (dat == 'S') stage = 1;
            else stage = 0;
            break;
        case 1:
            if (dat == 'T') stage = 2;
            else goto L_Check1st;
            break;
        case 2:
            if (dat == 'C') stage = 3;
            else goto L_Check1st;
            break;
        case 3:
            if (dat == 'I') stage = 4;
            else goto L_Check1st;
            break;
        case 4:
            if (dat == 'S') stage = 5;
            else goto L_Check1st;
            break;
        case 5:
            if (dat == 'P') stage = 6;
            else goto L_Check1st;
            break;
        case 6:
            if (dat == '$')                             //当检测到正确的用户下载命令时
                IAP_CONTR = 0x60;                       //复位到系统区
            else goto L_Check1st;
        break;
        }
    }
}

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;
```

STC MCU

```
    SCON = 0x50;                                //设置用户串口模式为8 位数据位
    TMOD = 0x00;
    AUXR = 0x40;
    TH1 = BR2400 >> 8;                          //设置串口波特率为115200
    TL1 = BR2400;
    TR1 = 1;
    ES = 1;
    EA = 1;

    stage = 0;

    while (1)
    {
        ...                                     //用户代码
    }
}
```

STC MCU

附录K 使用第三方MCU对STC8A/8F系列单片机进行ISP下载范例程序

C 语言代码

```
//注意:使用本代码对 STC8A/F 系列的单片机进行下载时,必须要执行了 Download 代码之后,
//才能给目标芯片上电,否则目标芯片将无法正确下载

#include "reg51.h"

typedef     bit             BOOL;
typedef     unsigned char   BYTE;
typedef     unsigned short  WORD;

//宏、常量定义
#define     FALSE           0
#define     TRUE            1
#define     LOBYTE(w)       ((BYTE)(WORD)(w))
#define     HIBYTE(w)       ((BYTE)((WORD)(w) >> 8))

#define     MINBAUD         2400L
#define     MAXBAUD         115200L

#define     FOSC            11059200L           //主控芯片工作频率
#define     BR(n)           (65536 - FOSC/4/(n))    //主控芯片串口波特率计算公式
#define     T1MS            (65536 - FOSC/1000)     //主控芯片 1ms 定时初值

#define     FUSER           24000000L           //STC8A/F 系列目标芯片工作频率
#define     RL(n)           (65536 - FUSER/4/(n))   //STC8A/F 系列目标芯片串口波特率计算公式

sfr         AUXR = 0x8e;
sfr         P3M1 = 0xB1;
sfr         P3M0 = 0xB2;

//变量定义
BOOL f1ms;                          //1ms 标志位
BOOL UartBusy;                      //串口发送忙标志位
BOOL UartReceived;                  //串口数据接收完成标志位
BYTE UartRecvStep;                  //串口数据接收控制
BYTE TimeOut;                       //串口通讯超时计数器
BYTE xdata TxBuffer[256];           //串口数据发送缓冲区
BYTE xdata RxBuffer[256];           //串口数据接收缓冲区
char code DEMO[256];                //演示代码数据

//函数声明
void Initial(void);
void DelayXms(WORD x);
BYTE UartSend(BYTE dat);
void CommInit(void);
void CommSend(BYTE size);
```

```
BOOL Download(BYTE *pdat, long size);

//主函数入口
void main(void)
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    Initial();
    if (Download(DEMO, 256))
    {
        //下载成功
        P3 = 0xff;
        DelayXms(500);
        P3 = 0x00;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0x00;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0x00;
        DelayXms(500);
        P3 = 0xff;
    }
    else
    {
        //下载失败
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
    }

    while (1);
}

//1ms 定时器中断服务程序
void tm0(void) interrupt 1
{
    static BYTE Counter100;

    f1ms = TRUE;
    if (Counter100-- == 0)
    {
        Counter100 = 100;
        if (TimeOut) TimeOut--;
    }
}
```

STC MCU

```
//串口中断服务程序
void uart(void) interrupt 4
{
    static WORD RecvSum;
    static BYTE RecvIndex;
    static BYTE RecvCount;
    BYTE dat;

    if (TI)
    {
        TI = 0;
        UartBusy = FALSE;
    }

    if (RI)
    {
        RI = 0;
        dat  = SBUF;
        switch (UartRecvStep)
        {
        case 1:
            if (dat != 0xb9) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 2:
            if (dat != 0x68) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 3:
            if (dat != 0x00) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 4:
            RecvSum = 0x68 + dat;
            RecvCount = dat - 6;
            RecvIndex = 0;
            UartRecvStep++;
            break;
        case 5:
            RecvSum += dat;
            RxBuffer[RecvIndex++] = dat;
            if (RecvIndex == RecvCount)     UartRecvStep++;
            break;
        case 6:
            if (dat != HIBYTE(RecvSum))     goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 7:
            if (dat != LOBYTE(RecvSum))     goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 8:
            if (dat != 0x16) goto L_CheckFirst;
            UartReceived = TRUE;
            UartRecvStep++;
            break;
L_CheckFirst:
        case 0:
```

```
            default:
                CommInit();
                UartRecvStep = (dat == 0x46       ? 1 : 0);
                break;
            }
        }
}

//系统初始化
void Initial(void)
{
        UartBusy = FALSE;

        SCON = 0xd0;                            //串口数据模式必须为8 位数据+1 位偶检验
        AUXR = 0xc0;
        TMOD = 0x00;
        TH0 = HIBYTE(T1MS);
        TL0 = LOBYTE(T1MS);
        TR0 = 1;
        TH1 = HIBYTE(BR(MINBAUD));
        TL1 = LOBYTE(BR(MINBAUD));
        TR1 = 1;
        ET0 = 1;
        ES = 1;
        EA = 1;
}

//Xms 延时程序
void DelayXms(WORD x)
{
        do
        {
            f1ms = FALSE;
            while (!f1ms);
        } while (x--);
}

//串口数据发送程序
BYTE UartSend(BYTE dat)
{
        while (UartBusy);

        UartBusy = TRUE;
        ACC = dat;
        TB8 = P;
        SBUF = ACC;

        return dat;
}

//串口通讯初始化
void CommInit(void)
{
        UartRecvStep = 0;
        TimeOut = 20;
        UartReceived = FALSE;
}

//发送串口通讯数据包
```

```
void CommSend(BYTE size)
{
	WORD sum;
	BYTE i;

	UartSend(0x46);
	UartSend(0xb9);
	UartSend(0x6a);
	UartSend(0x00);
	sum = size + 6 + 0x6a;
	UartSend(size + 6);
	for (i=0; i<size; i++)
	{
		sum  += UartSend(TxBuffer[i]);
	}
	UartSend(HIBYTE(sum));
	UartSend(LOBYTE(sum));
	UartSend(0x16);
	while (UartBusy);

	CommInit();
}

//对 STC8A/F 系列的芯片进行 ISP 下载程序
BOOL Download(BYTE *pdat, long size)
{
	BYTE arg;
	BYTE offset;
	BYTE cnt;
	WORD addr;

	//握手
	CommInit();
	while (1)
	{
		if (UartRecvStep == 0)
		{
			UartSend(0x7f);
			DelayXms(10);
		}
		if (UartReceived)
		{
			arg = RxBuffer[4];
			if (RxBuffer[0] == 0x50) break;
			return FALSE;
		}
	}

	//设置参数(设置从芯片使用最高的波特率以及擦除等待时间等参数)
	TxBuffer[0] = 0x01;
	TxBuffer[1] = arg;
	TxBuffer[2] = 0x40;
	TxBuffer[3] = HIBYTE(RL(MAXBAUD));
	TxBuffer[4] = LOBYTE(RL(MAXBAUD));
	TxBuffer[5] = 0x00;
	TxBuffer[6] = 0x00;
	TxBuffer[7] = 0x81;
	CommSend(8);
	while (1)
```

STC MCU

```
{
    if (TimeOut == 0) return FALSE;
    if (UartReceived)
    {
        if (RxBuffer[0] == 0x01) break;
        return FALSE;
    }
}

//准备
TH1 = HIBYTE(BR(MAXBAUD));
TL1 = LOBYTE(BR(MAXBAUD));
DelayXms(10);
TxBuffer[0] = 0x05;
TxBuffer[1] = 0x00;
TxBuffer[2] = 0x00;
TxBuffer[3] = 0x5a;
TxBuffer[4] = 0xa5;
CommSend(5);
while (1)
{
    if (TimeOut == 0) return FALSE;
    if (UartReceived)
    {
        if (RxBuffer[0] == 0x05) break;
        return FALSE;
    }
}

//擦除
DelayXms(10);
TxBuffer[0] = 0x03;
TxBuffer[1] = 0x00;
TxBuffer[2] = 0x00;
TxBuffer[3] = 0x5a;
TxBuffer[4] = 0xa5;
CommSend(5);
TimeOut = 100;
while (1)
{
    if (TimeOut == 0) return FALSE;
    if (UartReceived)
    {
        if (RxBuffer[0] == 0x03) break;
        return FALSE;
    }
}

//写用户代码
DelayXms(10);
addr = 0;
TxBuffer[0] = 0x22;
TxBuffer[3] = 0x5a;
TxBuffer[4] = 0xa5;
offset = 5;
while (addr < size)
{
    TxBuffer[1] = HIBYTE(addr);
    TxBuffer[2] = LOBYTE(addr);
```

STC MCU

```
        cnt    = 0;
        while (addr < size)
        {
            TxBuffer[cnt+offset] = pdat[addr];
            addr++;
            cnt++;
            if (cnt >= 128) break;
        }
        CommSend(cnt + offset);
        while (1)
        {
            if (TimeOut == 0) return FALSE;
            if (UartReceived)
            {
                if ((RxBuffer[0] == 0x02) && (RxBuffer[1] == 'T')) break;
                return FALSE;
            }
        }
        TxBuffer[0] = 0x02;
    }

    ////写硬件选项
    ////如果不需要修改硬件选项,此步骤可直接跳过,此时所有的硬件选项
    ////都维持不变,MCU 的频率为上一次所调节频率
    ////若写硬件选项,MCU 的内部 IRC 频率将被固定写为 24M,其他选项恢复为出厂设置
    ////建议:第一次使用 STC-ISP 下载软件将从芯片的硬件选项设置好
    ////以后再使用主芯片对从芯片下载程序时不写硬件选项
    //DelayXms(10);
    //for (cnt=0; cnt<128; cnt++)
    //{
    //    TxBuffer[cnt] = 0xff;
    //}
    //TxBuffer[0] = 0x04;
    //TxBuffer[1] = 0x00;
    //TxBuffer[2] = 0x00;
    //TxBuffer[3] = 0x5a;
    //TxBuffer[4] = 0xa5;
    //TxBuffer[33] = arg;
    //TxBuffer[34] = 0x00;
    //TxBuffer[35] = 0x01;
    //TxBuffer[41] = 0xbf;
    //TxBuffer[42] = 0xbd;                  //P5.4 为 IO 口
    ////TxBuffer[42] = 0xad;                //P5.4 为复位脚
    //TxBuffer[43] = 0xf7;
    //TxBuffer[44] = 0xff;
    //CommSend(45);
    //while (1)
    //{
    //    if (TimeOut == 0) return FALSE;
    //    if (UartReceived)
    //    {
    //        if ((RxBuffer[0] == 0x04) && (RxBuffer[1] == 'T')) break;
    //        return FALSE;
    //    }
    //}

    // 下载完成
    return TRUE;
}
```

STC MCU

```
char code DEMO[256] =
{
    0x80,0x00,0x75,0xB2,0xFF,0x75,0xB1,0x00,0x05,0xB0,0x11,0x0E,0x80,0xFA,0xD8,0xFE,
    0xD9,0xFC,0x22,
};
```

备注：用户若需要设置不同的工作频率，可参考 7.3.7 和 7.3.8 章的范例代码

STC MCU

附录L 使用第三方应用程序调用STC发布项目程序对单片机进行ISP下载

使用 STC 的 ISP 下载软件生成的发布项目程序为可执行的 EXE 格式文件，用户可直接双击发布的项目程序运行进行 ISP 下载，也可在第三方的应用程序中调用发布项目程序进行 ISP 下载。下面介绍两种调用的方法。

简单调用

在第三方应用程序中只是简单创建发布项目程序的进程，其他的所有下载操作均在发布项目程序中进行，第三方应用程序此时只需要等待发布项目程序操作完成后，清理现场即可。

VC 代码

```
BOOL IspProcess()
{
    //定义相关变量
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    CString path;

    //发布项目程序的完整路径
    path = _T("D:\\Work\\Upgrade.exe");

    //变量初始化
    memset(&si, 0, sizeof(STARTUPINFO));
    memset(&pi, 0, sizeof(PROCESS_INFORMATION));

    //设置启动变量
    si.cb = sizeof(STARTUPINFO);
    GetStartupInfo(&si);
    si.wShowWindow = SW_SHOWNORMAL;
    si.dwFlags = STARTF_USESHOWWINDOW;

    //创建发布项目程序进程
    if (CreateProcess(NULL, (LPTSTR)(LPCTSTR)path, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
    {
        //等待发布项目程序操作完成
        //由于此处会阻塞主进程，所以建议新建工作进程，在工作进程中进行等待
        WaitForSingleObject(pi.hProcess,INFINITE);

        //清理工作
        CloseHandle(pi.hThread);
        CloseHandle(pi.hProcess);

        return TRUE;
    }
    else
    {
        AfxMessageBox(_T("创建进程失败 !"));
```

```
            return FALSE;
        }
}
```

高级调用

在第三方应用程序创建发布项目程序的进程，并在第三方应用程序中进行包括选择串口、开始 ISP 编程、停振 ISP 编程以及关闭发布项目程序等的全部 ISP 下载操作，而不需要在发布项目程序中进行界面互动。

VC 代码

```
//定义回调函数参数的数据结构
struct CALLBACK_PARAM
{
    DWORD dwProcessId;                  //主进程 ID
    HWND hMainWnd;                      //主窗口句柄
};

//枚举窗口的回调函数，用于获取主窗口句柄
BOOL CALLBACK EnumWindowCallBack(HWND hWnd, LPARAM lParam)
{
    CALLBACK_PARAM *pcp = (CALLBACK_PARAM *)lParam;
    DWORD id;

    GetWindowThreadProcessId(hWnd, &id);
    if ((pcp->dwProcessId == id) && (GetParent(hWnd) == NULL))
    {
        pcp->hMainWnd = hWnd;

        return FALSE;
    }

    return TRUE;
}

BOOL IspProcess()
{
    //定义相关变量
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    CALLBACK_PARAM cp;
    CString path;

    //发布项目程序中部分控件的 ID
    const UINT ID_PROGRAM       = 1046;
    const UINT ID_STOP          = 1044;
    const UINT ID_COMPORT       = 1009;
    const UINT ID_PROGRESS      = 1044;

    //发布项目程序的完整路径
    path = _T("D:\\Work\\Upgrade.exe");

    //变量初始化
    memset(&si, 0, sizeof(STARTUPINFO));
    memset(&pi, 0, sizeof(PROCESS_INFORMATION));
    memset(&cp, 0, sizeof(CALLBACK_PARAM));
```

```
//设置启动变量
si.cb = sizeof(STARTUPINFO);
GetStartupInfo(&si);
si.wShowWindow = SW_SHOWNORMAL;                    //此处若设置为SW_HIDE,就不会显示发布项目程序
                                                   //的操作界面,所有的ISP 操作都可在后台进行
si.dwFlags = STARTF_USESHOWWINDOW;

//创建发布项目程序进程
if (CreateProcess(NULL, (LPTSTR)(LPCTSTR)path, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
{
    //等待发布项目程序进程初始化完成
    WaitForInputIdle(pi.hProcess, 5000);

    //获取发布项目程序的主窗口句柄
    cp.dwProcessId = pi.dwProcessId;
    cp.hMainWnd = NULL;
    EnumWindows(EnumWindowCallBack, (LPARAM)&cp);

    if (cp.hMainWnd != NULL)
    {
        HWND hProgram;
        HWND hStop;
        HWND hPort;

        //获取发布项目程序主窗口中部分控件句柄
        hProgram = ::GetDlgItem(cp.hMainWnd, ID_PROGRAM);
        hStop = ::GetDlgItem(cp.hMainWnd, ID_STOP);
        hPort = ::GetDlgItem(cp.hMainWnd, ID_COMPORT);

        //设置发布项目程序中的串口号, 第3 个参数为0:COM1, 1:COM2, 2:COM3, ...
        ::SendMessage(hPort, CB_SETCURSEL, 0, 0);

        //触发编程按钮开始ISP 编程
        ::SendMessage(hProgram, BM_CLICK, 0, 0);

        //等待编程完成,
        //由于此处会阻塞主进程，所以建议新建工作进程，在工作进程中进行等待
        while (!::IsWindowEnabled(hProgram));

        //编程完成后关闭发布项目程序
        ::SendMessage(cp.hMainWnd, WM_CLOSE, 0, 0);
    }

    //等待进程结束
    WaitForSingleObject(pi.hProcess,INFINITE);

    //清理工作
    CloseHandle(pi.hThread);
    CloseHandle(pi.hProcess);

    return TRUE;
}
else
{
    AfxMessageBox(_T("创建进程失败 !"));

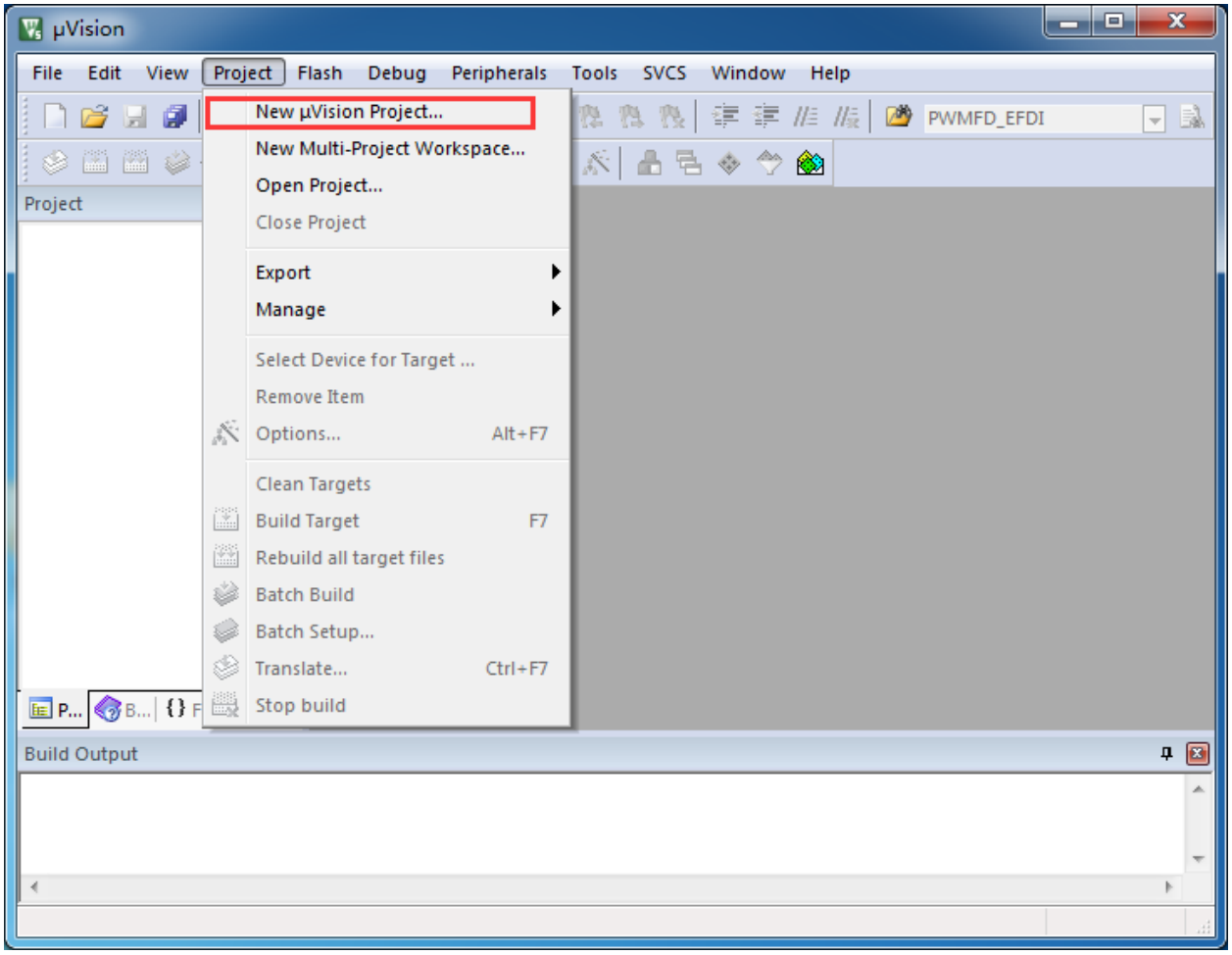
    return FALSE;
}
```

```
}
```

STC MCU

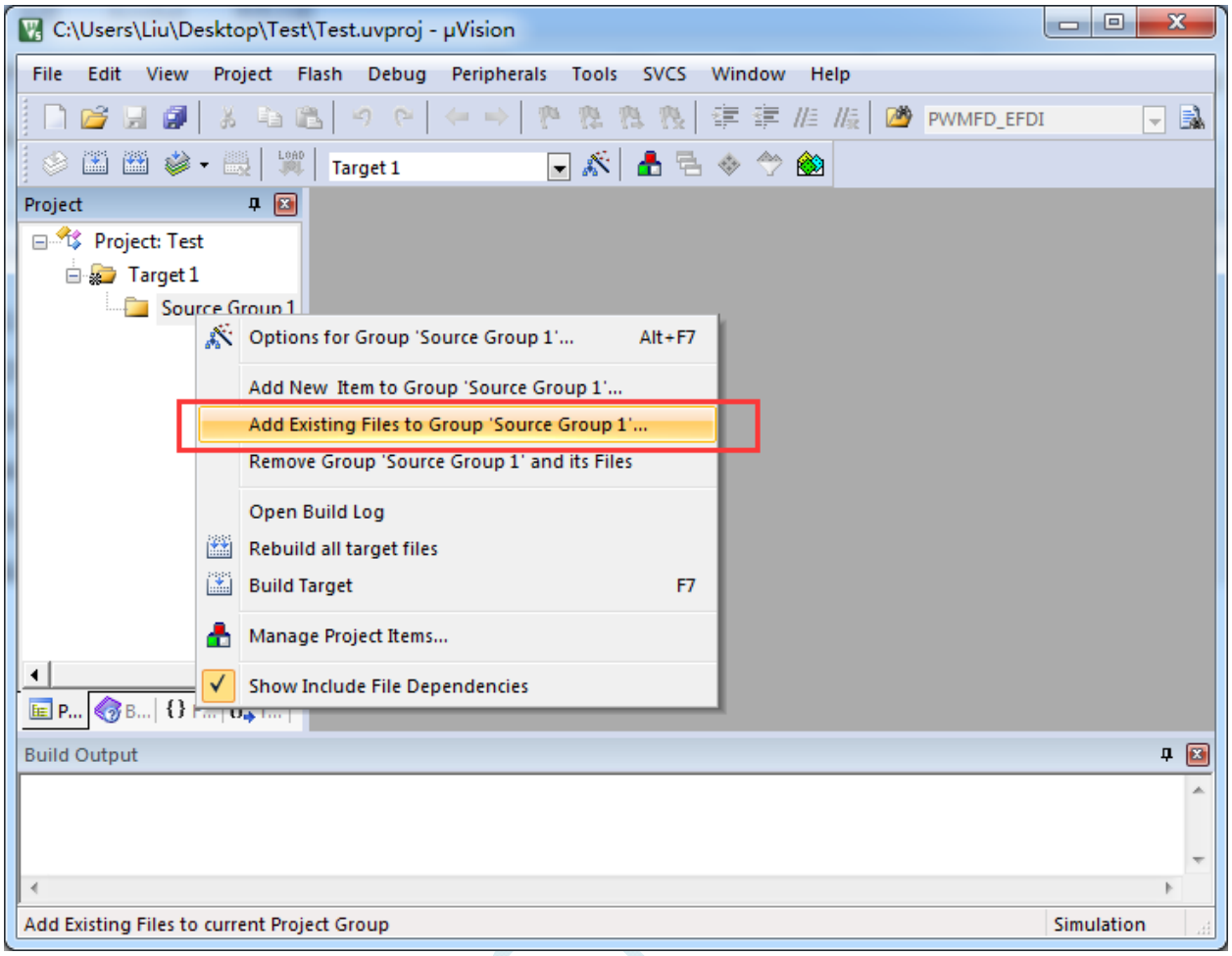
附录M 在Keil中建立多文件项目的方法

在 Keil 中，一般比较小的项目都只有一个源文件，但对于一些稍微复杂的项目往往需要多个源文件
建立多文件项目的方法如下：
1、首先打开 Keil，在菜单“Project”中选择“New uVision Project ...”

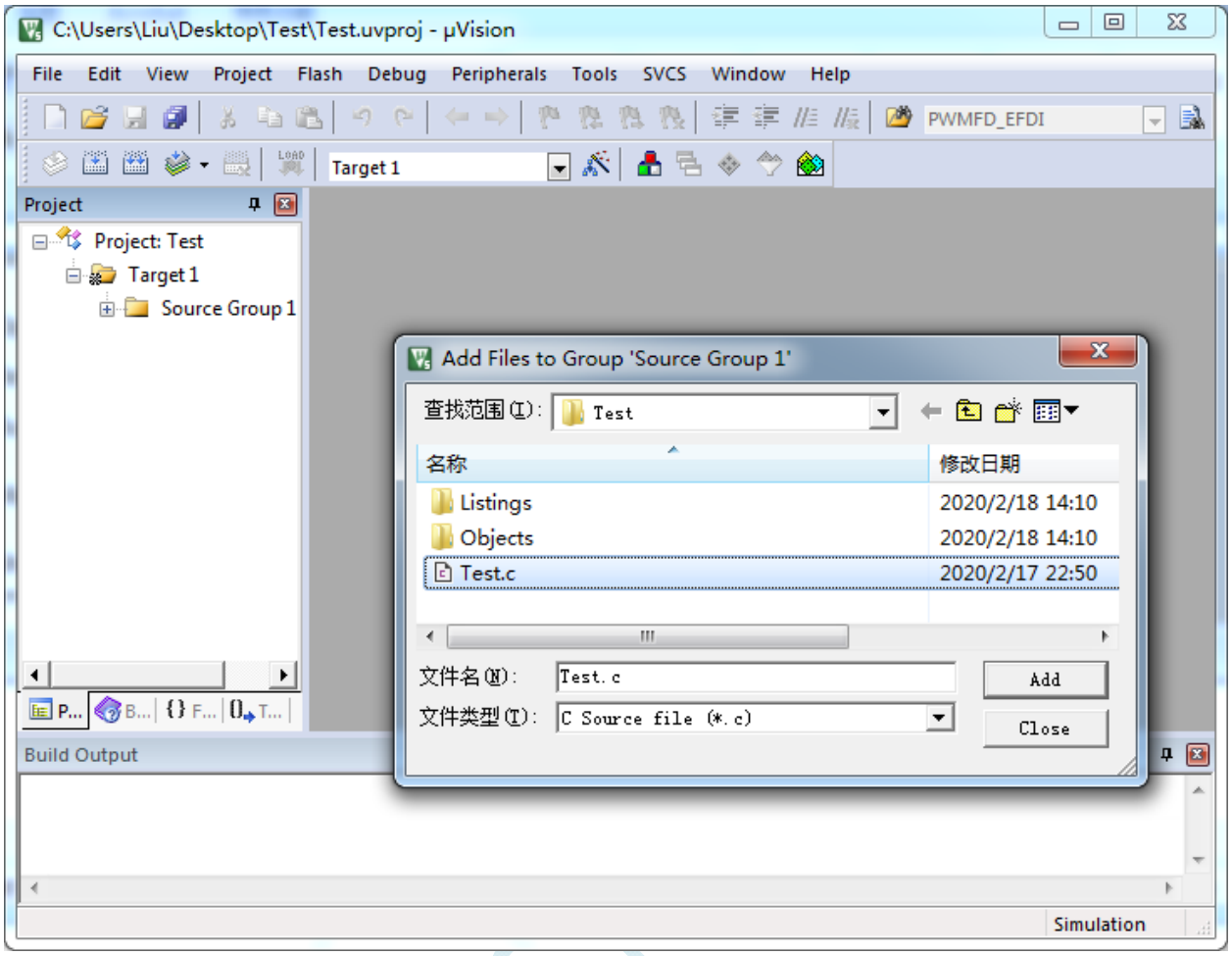


即可完成一个空项目的建立

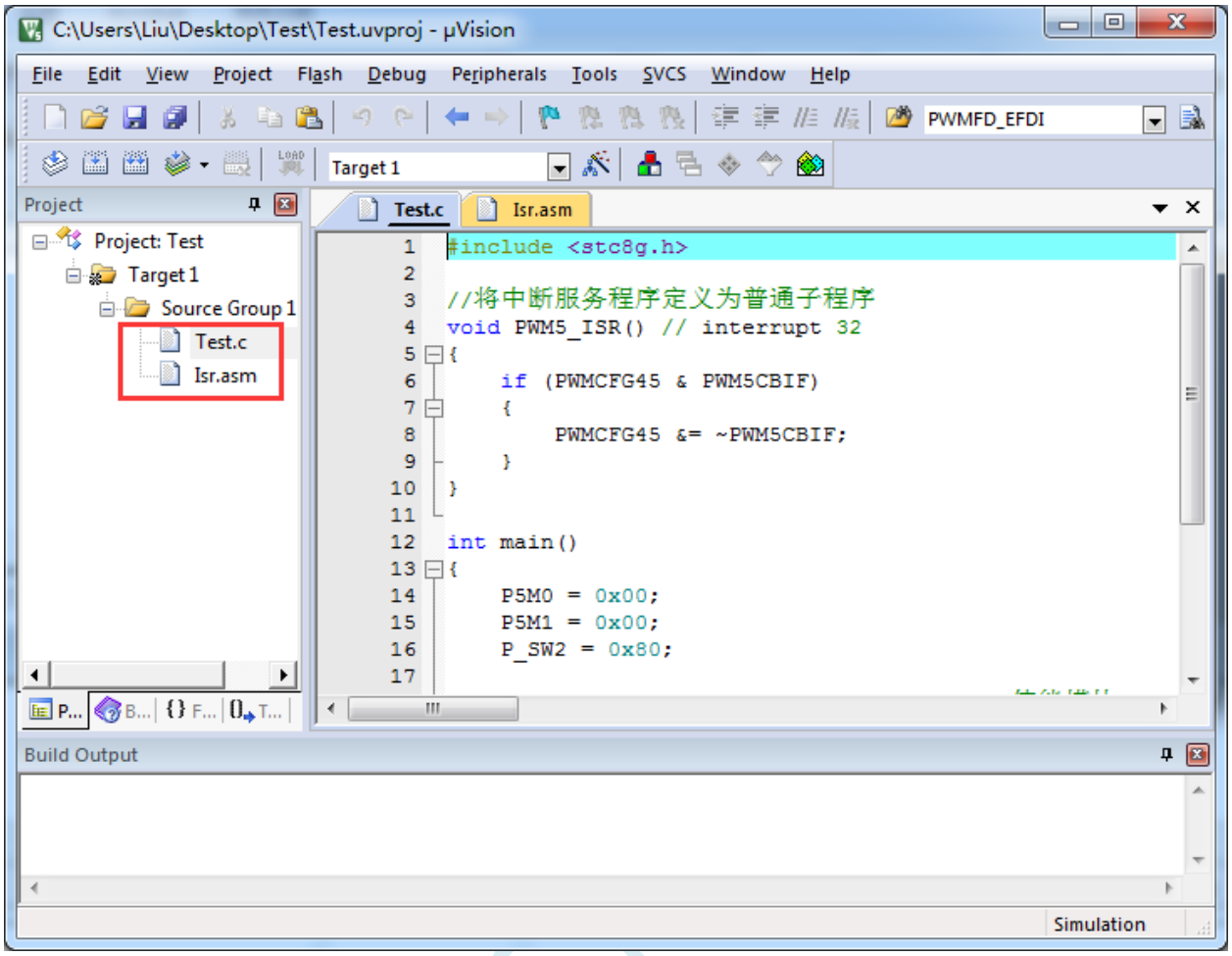
2、在空项目的项目树中，鼠标右键单击“Source Group 1”，并选择右键菜单中的“Add Existing Files to Group "Source Group 1" ...”



3、在弹出的文件对话框中，多次添加源文件



如下图所示即可完成多文件项目的建立


C:\Users\Liu\Desktop\Test\Test.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
PWMFD_EFDI
Target 1
Project
Project: Test
Target 1
Source Group 1
Test.c
Isr.asm
Test.c
Isr.asm
1 #include <stc8g.h>
2
3 //将中断服务程序定义为普通子程序
4 void PWM5_ISR() // interrupt 32
5 {
6 if (PWMCFG45 & PWM5CBIF)
7 {
8 PWMCFG45 &= ~PWM5CBIF;
9 }
10 }
11
12 int main()
13 {
14 P5M0 = 0x00;
15 P5M1 = 0x00;
16 P_SW2 = 0x80;
17
Build Output
Simulation

附录N 电气特性

N.1 绝对最大额定值

参数	最小值	最大值	单位	说明
存储温度	-55	+125	℃	
工作温度	-40	+85	℃	若工作温度高于 85℃（如 125℃附近），由于内部 IRC 时钟的频率在高温时的温漂大，建议使用外部高温时钟或或晶振。另外温度高时频率跑不快，如果必须使用内部 IRC 时钟，建议使用 24M 以下的工作频率；如果系统必须运行在较高频率，则请使用外部高可靠有源时钟。 若工作温度为-55℃附近，则工作电压不能太低，强烈建议 MCU-VCC 电压不要低于 3.0V，另外电源的上升速度也必须尽量快，最好能控制在毫秒级
工作电压	2.0	5.5	V	
VDD 对地电压	-0.3	+5.5	V	
I/O 口对地电压	-0.3	VDD+0.3	V	

STC MCU

N.2 直流特性（STC8A系列）

（VSS=0V，VDD=5.0V，测试温度=25℃）

标号	参数	范围				测试环境
		最小值	典型值	最大值	单位	
I_{PD}	掉电模式电流（SCC = 1）	-	0.08	-	uA	5.0V
	掉电模式电流（SCC = 0）	-	1.6	-	uA	5.0V
I_{WKT}	掉电唤醒定时器	-	5	-	uA	5.0V
I_{LVD}	低压检测模块	-	260	-	uA	5.0V
I_{IDL}	空闲模式电流（6MHz）	-	1.5	-	mA	5.0V
	空闲模式电流（11.0592MHz）		1.9		mA	5.0V
	空闲模式电流（20MHz）		2.7		mA	5.0V
	空闲模式电流（22.1184MHz）	-	3.0	-	mA	5.0V
	空闲模式电流（24MHz）	-	3.2	-	mA	5.0V
	空闲模式电流（内部 32KHz）	-	890	-	uA	5.0V
I_{NOR}	正常模式电流（6MHz）	-	3.0	-	mA	5.0V
	正常模式电流（11.0592MHz）	-	4.3	-	mA	5.0V
	正常模式电流（20MHz）	-	6.8	-	mA	5.0V
	正常模式电流（22.1184MHz）	-	7.4	-	mA	5.0V
	正常模式电流（24MHz）	-	7.8	-	mA	5.0V
	正常模式电流（内部 32KHz）	-	950	-	uA	5.0V
I_{CC}	普通工作模式电流	-	4	20	mA	5.0V
V_{IL1}	输入低电平	-	-	1.4	V	5.0V（打开施密特触发）
		-	-	1.5	V	5.0V（关闭施密特触发）
V_{IH1}	输入高电平（普通 I/O）	2.2	-	-	V	5.0V（打开施密特触发）
		1.6	-	-	V	5.0V（关闭施密特触发）
V_{IH2}	输入高电平（复位脚）	2.2	-	-	V	5.0V
I_{OL1}	输出低电平的灌电流	-	20	-	mA	5.0V，端口电压 0.45V
I_{OH1}	输出高电平电流（双向模式）	200	270	-	uA	5.0V
I_{OH2}	输出高电平电流（推挽模式）	-	20	-	mA	5.0V，端口电压 2.4V
I_{IL}	逻辑 0 输入电流	-	-	50	uA	5.0V，端口电压 0V
I_{TL}	逻辑 1 到 0 的转移电流	100	270	600	uA	5.0V，端口电压 2.0V
R_{PU}	IO 口上拉电阻	4.1	4.2	4.4	KΩ	5.0V
R_{PU}	IO 口上拉电阻	5.8	5.9	6.0	KΩ	3.3V

N.3 直流特性（STC8F系列）

（VSS=0V，VDD=5.0V，测试温度=25℃）

标号	参数	范围				测试环境
		最小值	典型值	最大值	单位	
I_{PD}	掉电模式电流（SCC = 1）	-	0.08	-	uA	5.0V
	掉电模式电流（SCC = 0）	-	1.5	-	uA	5.0V
I_{WKT}	掉电唤醒定时器	-	5	-	uA	5.0V
I_{LVD}	低压检测模块	-	260	-	uA	5.0V
I_{IDL}	空闲模式电流（6MHz）	-	1.3	-	mA	5.0V
	空闲模式电流（11.0592MHz）		1.7		mA	5.0V
	空闲模式电流（20MHz）		2.3		mA	5.0V
	空闲模式电流（22.1184MHz）	-	2.5	-	mA	5.0V
	空闲模式电流（24MHz）	-	2.6	-	mA	5.0V
	空闲模式电流（内部 32KHz）	-	850	-	uA	5.0V
I_{NOR}	正常模式电流（6MHz）	-	2.7	-	mA	5.0V
	正常模式电流（11.0592MHz）	-	3.8	-	mA	5.0V
	正常模式电流（20MHz）	-	5.9	-	mA	5.0V
	正常模式电流（22.1184MHz）	-	6.3	-	mA	5.0V
	正常模式电流（24MHz）	-	6.5	-	mA	5.0V
	正常模式电流（内部 32KHz）	-	950	-	uA	5.0V
I_{CC}	普通工作模式电流	-	4	20	mA	5.0V
V_{IL1}	输入低电平	-	-	1.4	V	5.0V（打开施密特触发）
		-	-	1.5	V	5.0V（关闭施密特触发）
V_{IH1}	输入高电平（普通 I/O）	1.7	-	-	V	5.0V（打开施密特触发）
		1.6	-	-	V	5.0V（关闭施密特触发）
V_{IH2}	输入高电平（复位脚）	1.6	-	1.7	V	5.0V
I_{OL1}	输出低电平的灌电流	-	20	-	mA	5.0V，端口电压 0.45V
I_{OH1}	输出高电平电流（双向模式）	200	270	-	uA	5.0V
I_{OH2}	输出高电平电流（推挽模式）	-	20	-	mA	5.0V，端口电压 2.4V
I_{IL}	逻辑 0 输入电流	-	-	50	uA	5.0V，端口电压 0V
I_{TL}	逻辑 1 到 0 的转移电流	100	270	600	uA	5.0V，端口电压 2.0V
R_{PU}	IO 口上拉电阻	4.1	4.2	4.4	KΩ	5.0V
R_{PU}	IO 口上拉电阻	5.8	5.9	6.0	KΩ	3.3V

N.4 内部IRC温漂特性（参考温度 25℃）

温度	范围		
	最小值	典型值	最大值
-40℃～85℃		−1.8%～+0.8%	
-20℃～65℃		−1.0%～+0.5%	

STC MCU

附录O 应用注意事项

O.1 关于STC8A系列芯片ADC的问题

对于 STC8A 系列的芯片，当用户使用 ADC_CONTR.7（ADC_POWER）寄存器打开 ADC 模块的电源时，ADC 模块会强制将 ADC_CONTR 寄存器中 ADC_CHS 所对于的 I/O 口设置为高阻输入模式，并关闭此端口上的数字输入通道。

例如：用户软件中执行完成“ADC_CONTR = 0x80”语句后，ADC 模块的电源被打开的同时，ADC 的通道 0 所对于的端口 P1.0 也会被强制设置为高阻输入模式，并关闭 P1.0 上所有的数字功能，包括 P1.0 口数字电平的输入输出、PWM0 的输出以及串口 2 的输入，所以此时 P1.0 口上将无法接收串口 2 的数据，也无法输出 PWM 波形。

STC8A 系列单片机的 15 路 ADC 所对应的 I/O 口均有多个复用的数字功能，所以为了避免打开 ADC 电源的同时影响不相关的 I/O 的数字功能，**软件中在设置 ADC_CONTR 寄存器时，务必将 ADC_POWER 和 ADC_CHS 同时设置**。比如用户系统中需要使用 P1.5 口进行 ADC 转换，则对 ADC_CONTR 进行初始化时请直接使用“ADC_CONTR = 0x85”语句进行初始化。

STC MCU

O.2 关于STC8A/8F系列芯片问题总结

1. 复位脚用于复位时，下拉电阻不大于 3K。在 1.5V 时有超过 200uA 的上拉电流，10K 电路拉不低。
2. 4 个串口用 9 位数据时，TB8 要装 2 次。
   （STC8A8K 系列的 G 版芯片无此问题、STC8A4K 系列的 G 版芯片无此问题、STC8F2K 系列 E 版芯片无此问题）
3. 使用 ADC 时，要注意下面的要求：AVCC 跟 VCC 电压差不要超过 0.1V，VREF 在 2.0~AVCC 之间，P1.0~P1.7 P0.0~P0.6 电压不能比 AVCC 高。
4. 使用外部晶振时，要注意下面的要求： AVCC 跟 VCC 电压差不要超过 0.1V，P1.0~P1.7 P0.0~P0.6 电压不能比 AVCC 高，否则晶振不起振，形同死机。
5. **STC8A/8F 系列的所有串口发送脚均不能设置为开漏模式，否则无法输出串口数据**
6. 4 个串口的发送脚 TXDn 要下面的三选一：
   A、设置为推挽输出，
   B、允许内部的上拉 3K7 电阻，
   C、外加 3K~5.1K 上拉电阻。
   （STC8A8K 系列的 G 版芯片无此问题、STC8A4K 系列的 G 版芯片无此问题、STC8F2K 系列 E 版芯片无此问题）
7. SPI 的 SCLK 和 MOSI 如果设置为开漏输出并且 IO 输出高，则 SCLK 和 MOSI 没有输出信号，但将这两个口输出低电平，则能正常输出。并且，这两个信号是推挽输出，与 IO 设置无关。
8. 8 通道增强型 PWM 的输出，切换到 P1.0~P1.7 或 P2.0~P2.7，IO 设置为准双向口或推挽输出，PWM 输出正常，均为推挽输出，并且再直接操作 IO 将不影响 PWM 波形。但是如果将 IO 设置为开漏输出（允许内部上拉电阻或外接上拉电阻），则如果对应的 IO 输出高电平，PWM 无输出（IO 为高阻），而对应的 IO 输出低电平，PWM 有推挽输出。
9. 4 通道 PCA 工作于 PWM 时，切换到 P1.4~P1.7 或 P2.3~P2.6 或 P3.3~P3.0，，情况同上类似，IO 设置为准双向口或推挽输出，PWM 输出正常，均为推挽输出，并且再直接操作 IO 将不影响 PWM 波形。但是如果将 IO 设置为开漏输出（允许内部上拉电阻或外接上拉电阻），则如果对应的 IO 输出高电平，PWM 无输出（IO 为高阻），而对应的 IO 输出低电平，PWM 有推挽输出。
10. 4 通道 PCA 工作于高速输出时，受到同一个端口其余 IO 翻转的影响，比如 PCA3 从 P2.6 高速输出脉冲，P2 口除了 P2.6 外任何一个 P2 口的改变都会影响到 P2.6 波形，哪怕是将 IO 设置为高阻。
11. 定时器 0 从 P3.5 高速输出、定时器 1 从 P3.4 高速输出、定时器 2 从 P1.3 高速输出、定时器 3 从 P0.5 高速输出、定时器 4 从 P0.7 高速输出，这些 IO 设置为准双向口或推挽输出，脉冲输出正常，均为推挽输出，并且再直接操作 IO 将不影响输出波形。但是如果将 IO 设置为开漏输出（允许内部上拉电阻或外接上拉电阻），则如果对应的 IO 输出高电平，波形无输出（IO 为高阻），而对应的 IO 输出低电平，波形有推挽输出。

O.3 关于使用CLR指令关闭EA的重要说明

对于 STC8 系列的 MCU，为了加快指令的执行速度，芯片内部对指令采用的是 4 级流水线的取指-解码-执行的并行体系，使得原本需要 12~48 个时钟周期才能完成的指令，在 STC8 系列的 MCU 中除了 MUL、DIV、DA、MOVC、MOVX 以及跳转指令外，其余指令均只需要一个时钟周期就能完成，注意此处所说的完成并不是真正的执行完成，而是在本时钟周期内完成对当前指令的解码后，在下一个时钟周期对下一条指令执行解码操作的同时执行上一条指令的动作。这样操作的好处是在宏观上确实加快了指令的执行速度，但因此而带来的负面影响是对部分指令的执行效果有一个时钟的迟滞现象。

前面所描述的迟滞现象对基于冯•洛伊曼体系结构的 8051 程序代码来说不会有任何问题，因为指令不会出现并发现象，但由于 8051 代码中存在中断机制，中断可能随时会打断当前顺序执行的代码，此时前面所描述的迟滞现象就可能会产生问题，所以一般的做法是当主循环中需要修改的变量可能会与中断中有访问冲突或者主循环中需要修改的变量在中断中需要进行逻辑判断时，就需要在主循环中在对这种变量进行修改前，先使用 CLR EA 指令将中断暂时关闭，等待对变量修改完成后在使用 SETB EA 指令打开中断，从而达到主循环和中断对变量互斥访问的目的。

注意：CLR EA 指令本身也有迟滞现象，即 CLR EA 之后，EA 并不是立即被关闭的，而是需要等待下一个时钟周期完成后 EA 才会被关闭，也就是说，运行完 CLR EA 后，需要再执行一条语句，EA 才会被真正关闭。所以正确的程序代码编写方式是在 CLR EA 指令后加 1~2 个 NOP 指令，因为 NOP 的执行对任何算式逻辑运算和逻辑判断都不会造成影响。

正确的关闭 EA 的汇编代码如下：

```
...
NOP
CLR     EA
NOP
...
NOP
SETB    EA
NOP
...
```

正确的关闭 EA 的 C 代码如下：

```
...
_nop_();
EA = 0;
_nop_();
...
_nop_();
EA = 1;
_nop_();
...
```

另外，对于 STC15 系列 MCU 也需要注意此问题，由于 STC15 系列的内部硬件也同样使用的是 4 级流水线的体系，所以也会有类似上面的问题。所以一般正确的做法也需要参考上面的示例代码来编写程序。

O.4 关于EEPROM编程和擦除等待时间的重要说明

表一（STC8A 系列和 STC8F 系列 EEPROM 操作时间需求）

EEPROM 操作	最短时间	最长时间
编程	6us	7.5us
擦除	4ms	6ms

表二（STC8A 系列和 STC8F 系列 EEPROM 操作相应等待参数的时间等待周期）

IAP_WT[2:0]			编程等待时钟数	擦除等待时钟数	适合的频率
1	1	1	7 个时钟	5000 个时钟	1MHz
1	1	0	14 个时钟	10000 个时钟	2MHz
1	0	1	21 个时钟	15000 个时钟	3MHz
1	0	0	42 个时钟	30000 个时钟	6MHz
0	1	1	84 个时钟	60000 个时钟	12MHz
0	1	0	140 个时钟	100000 个时钟	20MHz
0	0	1	168 个时钟	120000 个时钟	24MHz
0	0	0	301 个时钟	215000 个时钟	30MHz

STC8A 系列和 STC8F 系列 MCU 的内部 EEPROM 的编程和擦除等待时间必须达到表一中的要求，等待时间不可过短，也不可过长。

编程的等待时间必须在 6us～7.5us 之间，编程等待时间过小（小于最短时间 6us），则被编程的目标存储单元内部的数据可能不可靠（数据的保存期限可能达不到 25 年）；若等待时间过长（大于最长时间 7.5us 的 1.5 倍，即大于 11.25us），也可能由于有数据干扰而导致写入的数据不正确。在确保编程的等待时间要求，并在编程完成后进行数据读出对比校验，若校验正确，数据便编程正确了。

擦除的等待时间必须在 4ms～6ms 之间，擦除等待时间过小（小于最短时间 4ms），则被擦除的目标存储扇区可能没有被擦除干净；若等待时间过长（大于最长时间 6ms 的 1.5 倍，即大于 9ms），则会缩短 EEPROM 的使用寿命，即原本 10 万次的擦除寿命可能会缩短为 5 万次。

编程与擦除的等待时间请严格按照表二所给的推荐频率进行合适的选择，假如工作频率为 12MHz，请按照表二推荐将等待参数设置为 011B，若 CPU 实际的工作频率并不在表二所推荐的频率之列，则需要根据实际的频率以及表二中实际的等待时钟数进行计算，找出满足表一时间需求的等待时间参数。

例如：工作频率为 4MHz，若选择等待参数为 101B，则编程时间为 21/4MHz = 5.25us，擦除时间为 15000/4MHz = 3.75ms，时间明显不够，所以应该选择等待参数为 100B，则编程时间为 42/4MHz = 10.5us，擦除时间为 30000/4MHz = 7.5ms，时间均在最短时间和最长时间的 1.5 倍之间。

注意：EEPROM 等待操作的时钟是指对主时钟进行分频后的系统时钟，即 CPU 实际的工作时钟。若单片机使用的是内部高精度 IRC，则 EEPROM 等待操作的时钟为使用 ISP 下载软件下载时经过调节后的频率；若单片机使用的外部晶振，则 EEPROM 等待操作的时钟为外部晶振频率经过 CLKDIV 寄存器分频后的时钟（例如：若单片机使用外部晶振，且外部晶振的频率为 24MHz，CLKDIV 寄存器的值设置为 4，则 EEPROM 等待操作的时钟频率为 24MHz/4 = 6MHz，此时等待参数应选择 100B，而不能选择 001B）。

下表为之前版本资料错误的部分

IAP_WT[2:0]	读字节	~~写字节~~	~~擦除扇区~~	时钟频率

			（2 个时钟）	~~（约 55us）~~	~~（约 21ms）~~	
1	1	1	2 个时钟	~~55 个时钟~~	~~21012 个时钟~~	1MHz
1	1	0	2 个时钟	~~110 个时钟~~	~~42024 个时钟~~	2MHz
1	0	1	2 个时钟	~~165 个时钟~~	~~63036 个时钟~~	3MHz
1	0	0	2 个时钟	~~330 个时钟~~	~~126072 个时钟~~	6MHz
0	1	1	2 个时钟	~~660 个时钟~~	~~252144 个时钟~~	12MHz
0	1	0	2 个时钟	~~1100 个时钟~~	~~420240 个时钟~~	20MHz
0	0	1	2 个时钟	~~1320 个时钟~~	~~504288 个时钟~~	24MHz
0	0	0	2 个时钟	~~1760 个时钟~~	~~672384 个时钟~~	30MHz

下表为针对之前的错误修改正确后的参数

IAP_WT[2:0]			读字节 （2 个时钟）	**写字节 （约 6～7.5us）**	**擦除扇区 （约 4～6ms）**	时钟频率
1	1	1	2 个时钟	**7 个时钟**	**5000 个时钟**	1MHz
1	1	0	2 个时钟	**14 个时钟**	**10000 个时钟**	2MHz
1	0	1	2 个时钟	**21 个时钟**	**15000 个时钟**	3MHz
1	0	0	2 个时钟	**42 个时钟**	**30000 个时钟**	6MHz
0	1	1	2 个时钟	**84 个时钟**	**60000 个时钟**	12MHz
0	1	0	2 个时钟	**140 个时钟**	**100000 个时钟**	20MHz
0	0	1	2 个时钟	**168 个时钟**	**120000 个时钟**	24MHz
0	0	0	2 个时钟	**301 个时钟**	**215000 个时钟**	30MHz

O.5 STC8F2K64S4 系列应用注意事项

1. **STC8F2K64S4 系列 E 版芯片重要说明 1**
   修正了 D 版芯片中有关 I2C 的下列问题
   a. **使用 STC8F2K64S4 系列 E 版芯片作从机，当从机地址与本机不匹配时，MCU 不会返回应答信号（D 版芯片会错误的回 ACK）**
   b. **使用 STC8F2K64S4 系列 E 版芯片作从机，当主机完成数据读取后给出 NAK 应答信号时，MCU 不会将下一个数据位送到 SDA 总线上（对于 D 版芯片，无论主机回 ACK 还是 NAK，MCU 都会错误的将下一个数据位送到 SDA 总线上）**
   c. **使用 STC8F2K64S4 系列 E 版芯片作主机，当 I2C 总线上产生干扰信号时，I2C 通讯会因此而中断，此时只需要将 I2C 总使能位 ENI2C（I2CCFG 寄存器的 bit7）关闭，然后再打开，I2C 主机即可恢复正常工作状态（当发生此类情况时，D 版芯片必须给 MCU 重新上电才可用）**
2. **STC8F2K64S4 系列 E 版芯片重要说明 2**
   修正了 D 版芯片中关于串口 1 的模式 2 和模式 3 时,在设置发送数据的第 9 位（TB8）时，需要连续设置两次才有效的问题，E 版芯片只需要设置一次即可。
3. **STC8F2K64S4 系列 E 版芯片重要说明 3**
   修正了下列串口发送数据时需要加上拉电阻或者需要设置为强推挽模式的问题，E 版芯片下列串口发送脚在发送数据时不需要额外处理：
   a. **TXD（P3.1）、TXD_2（P3.7）、TXD_3（P1.7）、TXD_4（P4.4）**
   b. **TXD2（P1.1）、TXD2_2（P4.2）**
   c. **TXD3（P0.1）**
   d. **TXD4（P0.3）**
   e. **注意：目前仍有 TXD3_2（P5.1）和 TXD4_2（P5.3）作为串口发送数据时需要加上拉电阻或者需要设置为推挽模式**

4. STC8F2K64S4 系列 D 版芯片重要说明 1
   所有串口(包括串口 1、串口 2、串口 3、串口 4)的串口发送端发送串口数据时，发送端口均需要进行下面的设置：（3 种方式任选其一）
   a. 设置 I/O 口为准双向口模式并打开内部的上拉电阻
   b. 设置 I/O 口为准双向口模式并外接 3～10K 的上拉电阻
   c. 设置 I/O 口为强推挽模式
5. STC8F2K64S4 系列 D 版芯片重要说明 2
   串口 1 的模式 2 和模式 3 时,在设置发送数据的第 9 位（TB8）时，需要连续设置两次才有效。串口 2、串口 3 和串口 4 无此问题

O.6 STC8F2K64S2 系列应用注意事项

1. **STC8F2K64S2 系列 E 版芯片重要说明 1**
   修正了 D 版芯片中有关 I2C 的下列问题
   a. 使用 STC8F2K64S2 系列 E 版芯片作从机，当从机地址与本机不匹配时，MCU 不会返回应答信号（D 版芯片会错误的回 ACK）
   b. 使用 STC8F2K64S2 系列 E 版芯片作从机，当主机完成数据读取后给出 NAK 应答信号时，MCU 不会将下一个数据位送到 SDA 总线上（对于 D 版芯片，无论主机回 ACK 还是 NAK，MCU 都会错误的将下一个数据位送到 SDA 总线上）
   c. 使用 STC8F2K64S2 系列 E 版芯片作主机，当 I2C 总线上产生干扰信号时，I2C 通讯会因此而中断，此时只需要将 I2C 总使能位 ENI2C（I2CCFG 寄存器的 bit7）关闭，然后再打开，I2C 主机即可恢复正常工作状态（当发生此类情况时，D 版芯片必须给 MCU 重新上电才可用）
2. **STC8F2K64S2 系列 E 版芯片重要说明 2**
   修正了 D 版芯片中关于串口 1 的模式 2 和模式 3 时,在设置发送数据的第 9 位（TB8）时，需要连续设置两次才有效的问题，E 版芯片只需要设置一次即可。

3. STC8F2K64S2 系列 D 版芯片重要说明 1
   所有串口(包括串口 1、串口 2、串口 3、串口 4)的串口发送端发送串口数据时，发送端口均需要进行下面的设置：（3 种方式任选其一）
   a. 设置 I/O 口为准双向口模式并打开内部的上拉电阻
   b. 设置 I/O 口为准双向口模式并外接 3～10K 的上拉电阻
   c. 设置 I/O 口为强推挽模式
4. STC8F2K64S2 系列 D 版芯片重要说明 2
   串口 1 的模式 2 和模式 3 时,在设置发送数据的第 9 位（TB8）时，需要连续设置两次才有效。串口 2、串口 3 和串口 4 无此问题

O.7 STC8A8K64S4A12 系列应用注意事项

1. **STC8A8K64S4A12 系列 G 版芯片重要说明 1**
   修正了 F 版芯片中有关 I2C 的下列问题
   a. **使用 STC8A8K64S4A12 系列 G 版芯片作从机，当从机地址与本机不匹配时，MCU 不会返回应答信号（F 版芯片会错误的回 ACK）**
   b. **使用 STC8A8K64S4A12 系列 G 版芯片作从机，当主机完成数据读取后给出 NAK 应答信号时，MCU 不会将下一个数据位送到 SDA 总线上（对于 F 版芯片，无论主机回 ACK 还是 NAK，MCU 都会错误的将下一个数据位送到 SDA 总线上）**
   c. **使用 STC8A8K64S4A12 系列 G 版芯片作主机，当 I2C 总线上产生干扰信号时，I2C 通讯会因此而中断，此时只需要将 I2C 总使能位 ENI2C（I2CCFG 寄存器的 bit7）关闭，然后再打开，I2C 主机即可恢复正常工作状态（当发生此类情况时，F 版芯片必须给 MCU 重新上电才可用）**
2. **STC8A8K64S4A12 系列 G 版芯片重要说明 2**
   修正了 F 版芯片中关于串口 1 的模式 2 和模式 3 时,在设置发送数据的第 9 位（TB8）时，需要连续设置两次才有效的问题，G 版芯片只需要设置一次即可。
3. **STC8A8K64S4A12 系列 G 版芯片重要说明 3**
   修正了下列串口发送数据时需要加上拉电阻或者需要设置为强推挽模式的问题，G 版芯片下列串口发送脚在发送数据时不需要额外处理：
   a. **TXD（P3.1）、TXD_3（P1.7）**
   b. **TXD2（P1.1）、TXD2_2（P4.2）**
   c. **注意：目前仍有 TXD_2（P3.7）、TXD_4（P4.4）、TXD3（P0.1）、TXD3_2（P5.1）、TXD4（P0.3）和 TXD4_2（P5.3）作为串口发送数据时需要加上拉电阻或者需要设置为推挽模式**
4. **增强型 PWM 的中断请求标志位修改为：当发生中断请求时硬件自动写 1 置中断请求标志，用户只能写 0 清中断请求标志（用户程序中不能写 1 设置中断请求标志，用户写 1 时视为无效）**

5. STC8A8K64S4A12 系列 F 版芯片重要说明 1
   所有串口(包括串口 1、串口 2、串口 3、串口 4)的串口发送端发送串口数据时，发送端口均需要进行下面的设置：（3 种方式任选其一）
   a. 设置 I/O 口为准双向口模式并打开内部的上拉电阻
   b. 设置 I/O 口为准双向口模式并外接 3～10K 的上拉电阻
   c. 设置 I/O 口为强推挽模式
6. STC8A8K64S4A12 系列 F 版芯片重要说明 2
   串口 1 的模式 2 和模式 3 时,在设置发送数据的第 9 位（TB8）时，需要连续设置两次才有效。串口 2、串口 3 和串口 4 无此问题

O.8 STC8A4K64S2A12 系列应用注意事项

1. **STC8A4K64S2A12 系列 G 版芯片重要说明 1**
   修正了 F 版芯片中有关 I2C 的下列问题
   a. **使用 STC8A4K64S2A12 系列 G 版芯片作从机，当从机地址与本机不匹配时，MCU 不会返回应答信号（F 版芯片会错误的回 ACK）**
   b. **使用 STC8A4K64S2A12 系列 G 版芯片作从机，当主机完成数据读取后给出 NAK 应答信号时，MCU 不会将下一个数据位送到 SDA 总线上（对于 F 版芯片，无论主机回 ACK 还是 NAK，MCU 都会错误的将下一个数据位送到 SDA 总线上）**
   c. **使用 STC8A4K64S2A12 系列 G 版芯片作主机，当 I2C 总线上产生干扰信号时，I2C 通讯会因此而中断，此时只需要将 I2C 总使能位 ENI2C（I2CCFG 寄存器的 bit7）关闭，然后再打开，I2C 主机即可恢复正常工作状态（当发生此类情况时，F 版芯片必须给 MCU 重新上电才可用）**
2. **STC8A4K64S2A12 系列 G 版芯片重要说明 2**
   修正了 F 版芯片中关于串口 1 的模式 2 和模式 3 时,在设置发送数据的第 9 位（TB8）时，需要连续设置两次才有效的问题，G 版芯片只需要设置一次即可。
3. **STC8A4K64S2A12 系列 G 版芯片重要说明 3**
   修正了下列串口发送数据时需要加上拉电阻或者需要设置为强推挽模式的问题，G 版芯片下列串口发送脚在发送数据时不需要额外处理：
   a. **TXD（P3.1）、TXD_3（P1.7）**
   b. **TXD2（P1.1）、TXD2_2（P4.2）**
   c. **注意：目前仍有 TXD_2（P3.7）、TXD_4（P4.4）作为串口发送数据时需要加上拉电阻或者需要设置为推挽模式**
4. **增强型 PWM 的中断请求标志位修改为：当发生中断请求时硬件自动写 1 置中断请求标志，用户只能写 0 清中断请求标志（用户程序中不能写 1 设置中断请求标志，用户写 1 时视为无效）**

5. STC8A4K64S2A12 系列 F 版芯片重要说明 1
   所有串口(包括串口 1、串口 2、串口 3、串口 4)的串口发送端发送串口数据时，发送端口均需要进行下面的设置：（3 种方式任选其一）
   a. 设置 I/O 口为准双向口模式并打开内部的上拉电阻
   b. 设置 I/O 口为准双向口模式并外接 3～10K 的上拉电阻
   c. 设置 I/O 口为强推挽模式
6. STC8A4K64S2A12 系列 F 版芯片重要说明 2
   串口 1 的模式 2 和模式 3 时,在设置发送数据的第 9 位（TB8）时，需要连续设置两次才有效。串口 2、串口 3 和串口 4 无此问题

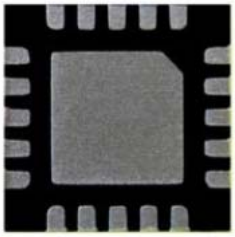
O.9 使用外部晶振对STC8 系列进行仿真的注意事项

1. 当使用 STC8 系列芯片作仿真器进行仿真时，若需要在用户系统中使用外部晶振，则在制作仿真芯片时，必须将内部 IRC 的频率设置为外部晶振相同的频率。
（例如：用户需要在系统中使用外部 11.0592MHz 晶振，则在制作仿真芯片时，需要将内部 IRC 的频率由默认的 24MHz 改为 11.0592MHz，否则仿真时会出错）

STC MCU

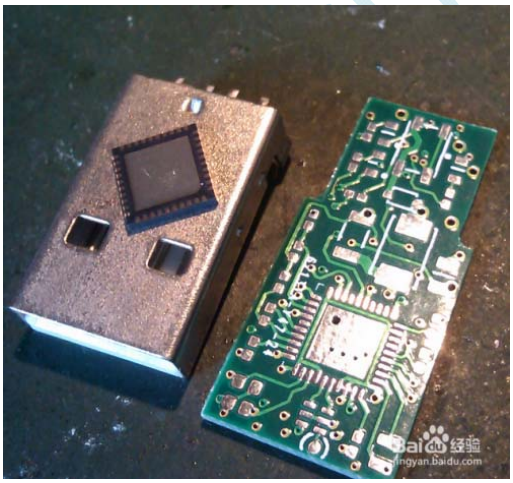
附录P QFN/DFN封装元器件焊接方法

STC 产品的封装形式中，增加了现在比较流行的 QFN 和 DFN 的封装。由于这种封装形式的芯片芯片的管脚在芯片底部，手工焊接有一定的难度。市面上有专门做工程样品焊接的小公司，可承接工程样品打样。如用户需要自行焊接，可参考下面的焊接方法。



1、 首先需要准备如下工具：电烙铁、热风枪、镊子、固定架等工具

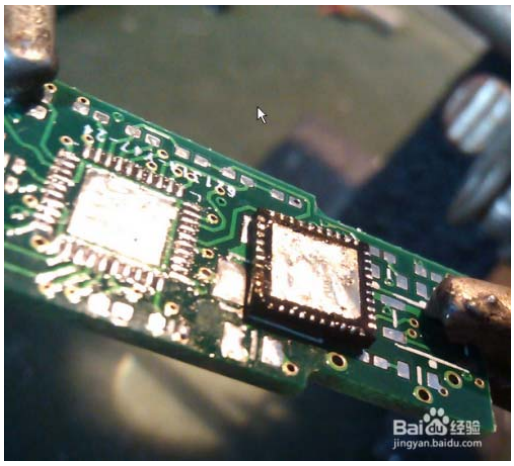
2、 需要焊接的 PCB 板和芯片如下图：



3、 先给板上芯片的焊盘上锡：



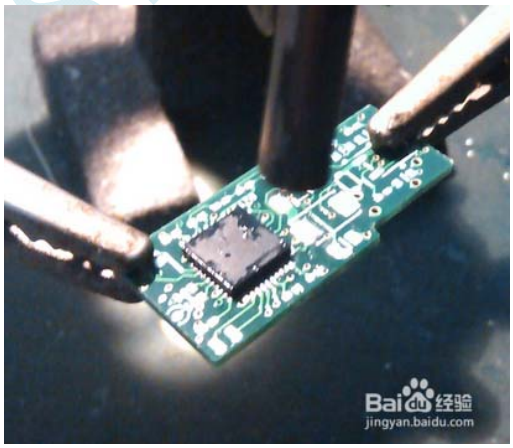
4、 然后给芯片底部上锡，这个上完锡后要弄平，尽量减少锡，但不能没有。



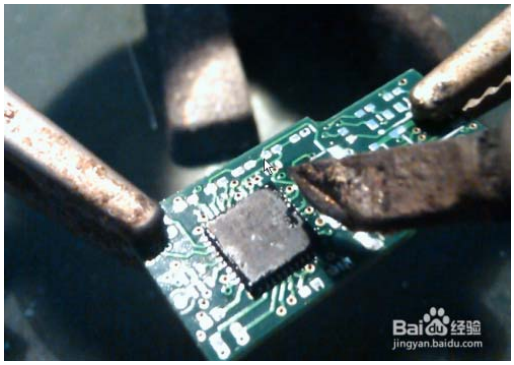
5、 调整热风枪温度，实际出风大概在 240 度左右，因为风枪质量不一样，根据实际情况调节。



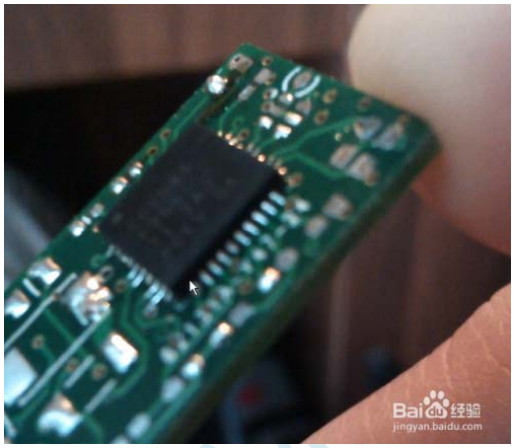
6、 把芯片放到焊盘上，一定要放正，然后用热风枪对着它吹，速度要均匀，直到锡溶化，一般 20 秒内。

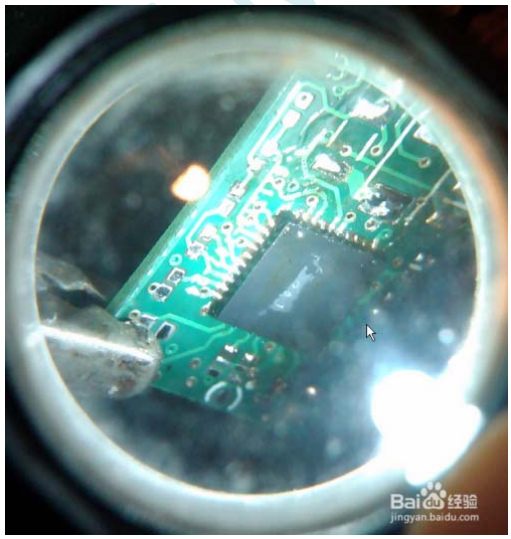


7、 用烙铁给芯片侧引脚上锡


Bai经验
jingyan.baidu.com

8、 焊接完成后的效果


Bai经验
jingyan.baidu.com


Bai经验
jingyan.baidu.com

附录Q 更新记录

- **2020/9/23**

1. 在定时器、串口、SPI、PCA、PWM 章节开头增加 I/O 说明
2. 更新 STC8A 系列应用注意事项

- **2020/9/4**

3. 更新 STC8A 系列应用注意事项
4. ADC 章节增加特别注意事项
5. 修改 ADC 转换速度计算公式

- **2019/8/15**

1. 修改范例中的错误

- **2019/5/29**

1. 增加使用第三方 MCU 对 STC8H 单片机进行 ISP 下载的示例代码
2. 增加使用 IO 口直接驱动 LCD 的示例代码

- **2018/11/16**

1. 在应用注意事项中总结 STC8A 系列芯片和 STC8F 系列芯片在使用中的一些注意事项
2. 删除 STC8 系列中串口中继功能的说明部分（STC8 系列不支持串口中继功能）
3. 增加 Win10 系统中 STC-USB 驱动的安装方法

- **2018/8/14**

1. 增加 STC8 系列 QFN64 和 QFN48 的封装尺寸图

- **2018/6/26**

1. 增加 STC8 系列 EEPROM 大小以及地址的说明
2. 增加使用 MOVC 读取 EEPROM 的范例程序

- **2018/5/7**

1. 增加使用 CLR EA 指令屏蔽中断的重要说明

STCMCU

2018/4/28

1. 增加 STC8F2K64S4 型号 E 版芯片的重要说明
2. 增加 STC8F2K64S2 型号 E 版芯片的重要说明
3. 增加 STC8A8K64S4A12 型号 G 版芯片的重要说明
4. 增加 STC8A4K64S2A12 型号 G 版芯片的重要说明

2018/4/25

1. 增加使用 PL2303-GL 进行 ISP 下载的参考线路图
2. 增加 STC8C 系列的提前通知和管脚图
3. 增加 STC8P 系列的提前通知和管脚图
4. 增加 STC8 系列使用外部晶振进行仿真的重要说明
5. 增加 STC8 系列内部 IRC 频率调整机制的说明章节
6. 增加 STC15 系列内部 IRC 频率调整机制的说明章节

2018/3/23

1. 修改 STC8F1K08S2 芯片 SOP16 管脚名称错误

2018/3/20

1. 增加从 ROM 和 RAM 中读取重要测试参数的范例程序
   a) 从存储器中读取 Bandgap 电压值
   b) 从存储器中读取全球唯一 ID 号
   c) 从存储器中读取 32K 掉电唤醒定时器频率
   d) 从存储器中读取 IRC 参数手动设置内部 IRC 频率

2018/3/13

1. 增加 STC8F1K08S2 型号 SOP16 的管脚图

2018/3/6

1. 增加利用比较器作掉电检测的范例程序
2. 增加利用比较器检测工作电压（电池电压）的范例程序
3. 增加利用芯片内部的 LVD（低电压检测）检测工作电压（电池电压）的范例程序
4. 增加 STC8F1K08S2 的特性说明和管脚图
5. 增加 STC8H1K08S2A10 的特性说明和管脚图

2018/1/30

1. 去掉范例程序中"using、_at_"的使用
2. 对 P2.0 的 RSTV 功能进行说明
3. 注明内部参考电压的电压值
4. 调整部分参考线路图

- **2017/11/27**

1. 增加掉电唤醒 IO 口的说明，增加掉电唤醒范例程序

- **2017/11/7**

1. 修正主时钟分频输出的时钟为经 CLKDIV 分频后的系统时钟（**之前错误的描述为主时钟分频输出的时钟为 CLKDIV 分频之前的主时钟**）
2. STC8 系列的比较器中断可设置 4 级中断优先级（之前的版本的资料描述有误）
3. 增加中断系统的结构框图
4. 增加 ADC 的第 16 通道测试外部电池电压的测试范例代码
5. 增加读取 STC8 系列 MCU 内部的重要参数的范例代码

- **2017/11/2**

1. 增加 STC8H1K08S2A10 系列和 STC8H1K08S2 系列的 SOP16 管脚图
2. 修正 EEPROM 编程和擦除时钟数
3. 增加 STC8F 系列和 STC8A 系列 EEPROM 编程和擦除等待时间设置的重要说明

- **2017/10/31**

1. 更新芯片封装图
2. 更新 ADC 典型应用线路图

- **2017/8/9**

1. 修正 STC8A4K64S2A12 系列的 IO 口数量为 59 个
2. 更新选型价格表

- **2017/8/1**

1. 更新应用注意事项

- **2017/7/27**

1. 更新选型及价格表

- **2017/7/12**

1. 增加 STC8F2K64S4 系列 D 版芯片的重要说明
2. STC8F2K64S4 系列 D 版芯片开始送样（可选封装 LQFP44、LQFP32、PDIP40）

- **2017/7/3**

1. STC8F2K 系列增加 TSSOP20 和 SOP16 的管脚图

STC MCU

2017/6/30

1. 增加 QFN32、TSSOP20、SOP16 的封装尺寸图
2. 更新代修正的重要说明

2017/5/17

1. 增加 STC8F2K64S2 系列
2. 增加 STC8A4K64S2A12 系列
3. 增加 STC8F1K08S2A10 系列
4. 增加 STC8F1K08S2 系列

2017/5/12

1. 增加了 I2C 主模式的辅助命令的说明（辅助命令仅对 STC8F2K64S4 系列的 C 版芯片、STC8A8K64S4A12 系列的 E 版芯片有效）
2. 增加了使用 U8W、U8-Mini 和 PL2303 进行 ISP 下载的参考线路图
3. 增加了端口内部上拉电阻和施密特触发控制的说明
4. 更新了 STC8 系列单片机的选型以及参考价格

2017/3/1

1. 增加重要说明章节以及仿真器使用的章节

2016/12/22

1. STC8F2K64S4 系列 C 版芯片无 PCA/CCP/PWM 功能
2. STC8A8K64S4A12 系列 B 版芯片有下列几点需要注意
   e) 串口 1 若使用定时器 1 的模式 2 做波特率发生器时，SMOD（PCON.7）位必须使能
   f) 所有的串口（包括串口 1、串口 2、串口 3 和串口 4）在接收数据时，都必须是两位停止位，否则可能会出现数据丢失
   g) 在访问内部扩展 XSFR 时，需要将 EXTRAM（AUXR.1）位置 1，否则在写入 XSFR 时会影响内部 XRAM 的最后 512 字节

2016/11/22

1. 增加范例程序
2. STC8F2K64S4 系列 B 版芯片有下列几点需要注意
   a) 串口 1 若使用定时器 1 的模式 2 做波特率发生器时，SMOD（PCON.7）位必须使能
   b) 所有的串口（包括串口 1、串口 2、串口 3 和串口 4）在接收数据时，都必须是两位停止位，否则可能会出现数据丢失
   c) 在访问内部扩展 XSFR 时，需要将 EXTRAM（AUXR.1）位置 1，否则在写入 XSFR 时会

影响内部 XRAM 的最后 512 字节

- **2016/9/13**

1. 增加 CAN 总线功能（定义 CAN 的管脚、CAN 中断相关 SFR）
   CAN 总线的功能 SFR 需要讨论后进行定义
2. 增加 IP3 和 IP3H，用于设置串口 3 和串口 4 的中断优先级
3. **上面两项功能处于规划阶段，目前芯片并无此功能**

- **2016/5/6**

1. 选型价格表中增加 I^2C 选项
2. 修正概述中关于串口 4 的端口切换的描述错误

- **2016/4/27**

1. 修改指令表中部分错误的指令执行时间

- **2016/4/22**

1. 修改 STC8A8K64S4 系列的 LQFP64S 的封装图和管脚排列
2. 修改 STC8F8K64S4 系列的 LQFP64S 的封装图和管脚排列

- **2016/4/15**

1. 修改 STC8A8K64S4 系列的 LQFP44、LQFP48 的封装图和管脚排列
2. 修改 STC8F8K64S4 系列的 LQFP44、LQFP48 的封装图和管脚排列
3. 修改 STC8F2K64S4 系列的 LQFP44 的封装图和管脚排列

STC MCU

附录R 官方网址说明

目前STC公司的高速、稳定的官方网址为：www.STCMCUDATA.com。

而旧的网址 www.STCMCU.com，由于服务器不稳定，网速慢等因素，所以建议大家不要使用。

STC MCU